caCORE 1.0

Technical Guide

NATIONAL® CANCER INSTITUTE

Center for Bioinformatics

NATIONAL INSTITUTES OF HEALTH

U.S. Department of
Health and Human Services

National Cancer Institute Center for Bioinformatics
6116 Executive Blvd. Suite 403
Rockville, Maryland 20852
USA

http://ncicb.nci.nih.gov
ncicb@nih.gov

TABLE OF CONTENTS

List of Figures

List of Tables

Introduction to caCORE: the NCICB Core Infrastructure

The last decade has produced a cornucopia of genomic information that has just begun to be examined. With this accumulation of bioinformatic data has come a paradigm shift to translational research, and a directive to more quickly advance discoveries in basic research to complex clinical settings and trials. This calls not only for advanced analytic tools and customized bioinformatics data warehouses, but, in addition, for computational environments and software tools that support the development of advanced data-mining applications and information management tasks in which the new scientist must participate. The National Cancer Institute's Center for Bioinformatics (NCICB) has as its mission the goal of bridging these diverse initiatives via a core infrastructure called caCORE.

Towards this end, NCICB is working to develop a "standards stack" within the cancer research community that integrates:

- controlled vocabularies (dictionaries, ontologies, and thesauri),
- common data elements (metadata), and
- logical models of entities within and across each domain.

We refer to the combination of technologies in the stack as caCORE. The caCORE infrastructure is composed of three primary components: the Enterprise Vocabulary Services (EVS), the Cancer Data Standards Repository (caDSR), and the Cancer Bioinformatics Infrastructure Objects (caBIO).

A guiding principle throughout all of the NCICB projects is the need to establish and/or adhere to agreed-upon standards of data representation, exchange, and manipulation. The ever-present need for well-defined terminologies in scientific domains has become even more critical with the increased prevalence of electronic data processing and exchange. The EVS provides a set of standardized, controlled vocabularies for the life sciences, along with tools and guidelines for the development and curation of such vocabularies. The vocabularies and ontologies managed by the EVS span multiple disciplines and domains, including human and mouse pathology, epidemiology, molecular biology, genetics, clinical trials, patient care, and various other biomedical and bioinformatic application areas. The EVS is described in more detail in Section 2 of this manual.

The caDSR addresses a related but somewhat orthogonal aspect of data representation and exchange; specifically, the need to standardize the terminology, report forms, and protocols implemented in clinical trials. Although much data have accrued over the years in ongoing clinical trials, to date, little effort has been made to standardize the methods of record-keeping and reporting. As a result, an enormous amount of valuable information that could be used to advance efforts in related studies has become effectively inaccessible, and the capacity to generalize important results from these legacy data has been precluded.

Based on the ISO/IEC11179 standard for metadata, the caDSR manages the NCI Common Data Elements (CDEs), and provides a registry for agreed upon clinical terms and their usage. This registry is implemented in Oracle 8i databases, as described in Section 3.

While the EVS and the caDSR address the representational needs and standardization issues involved in controlled vocabularies, report forms, and terminologies, the caBIO project provides a comprehensive set of pre-defined data structures, programming interfaces, and customized data sources to support the development of advanced software applications seeking to elucidate the molecular basis of cancer.

In keeping with the principle of conformance to emerging standards for data exchange, all of the caBIO data objects are "XML aware," and their design embodies many of the principles advocated by the Life Sciences Research Group at the Object Management Group (OMG). Several transparent programming interfaces are available, which support whatever might be the developer's language of choice — including Java, Perl, C++, Python, or even HTML.

Many of the data structures and development tools provided by caBIO were initially developed in response to the need to directly access information provided by the Cancer Genome Anatomy Project (CGAP) website. CGAP is an interactive website providing access to vast reserves of genomic information filtered by tissue type, histological status, chromosome location, and biological pathways. Some example applications that have used the CGAP resources include:

- Analysis of correlations between allelic variants of genes and disease states
- Identification of single-nucleotide polymorphisms from EST chromatograms
- Identification of potential tumor markers and antigens
- *In silico* cloning of novel endothelial-specific genes
- Clustering of highly expressed genes in chromosomal domains

The overwhelming success of CGAP and the invaluable resources it provides led to a good deal of "screen scraping" of the pages it produced. The information produced by a first query might serve as input to a second stage of analysis, and thus the application developer was forced to work around the HTML headers and banners to extract interim results.

But caBIO is more than a programmatic interface to CGAP; it provides access to many other data sources, as well as to software development tools that are customized for bioinformatic data-mining applications. One such application is NCI's Cancer Molecular Analysis Project (CMAP), which enables researchers to identify and evaluate molecular targets in cancer. The CMAP website was developed using the data structures and software tools provided by the caBIO infrastructure.

caBIO provides domain objects (Genes, Chromosomes, Sequences, etc.) which, in conjunction with *search criteria* objects, encapsulate the complexities of cross-platform data exchange and SQL query statements. Some of the data sources caBIO supplies access to include NCI's CGAP, CMAP, and GAI databases; NCBI's Unigene, Homolgene, and LocusLink databases; the Distributed Annotation Server (DAS) at UCSC; and BioCarta Pathway data.

In summary, the caCORE infrastructure brings a set of bridging technologies to the frontiers of cancer research. The EVS provides a web interface to a MetaThesaurus spanning over 70 controlled vocabularies specific to the areas of cancer resarch, prevention, and treatment. The caDSR provides a platform for registered common data elements to be used in the development of protocols, adverse event reports, and clinical report forms for use in clinical trials. The caBIO software development tools provide domain modeling of both the bioinformatic as well as administrative components of these efforts, and supply access to both customized data warehouses and public databases.

The remainder of this manual is organized as follows. The Developer's Guide to caBIO (Section 1) includes a description of the Unified Modeling Language (UML) model of the caBIO objects, an overview of the caBIO architecture and its underlying database search paradigm, setup instructions and examples for using the different caBIO application programming interfaces (APIs), and a description of the data sources to which caBIO provides access.

The discussion of the Enterprise Vocabulary Services in Section 2 includes a more detailed description of those services, an introduction to the EVS Metaphrase web server, instructions on how to use the EVS's Java API for programmatic access, and discussions of some of the more important local vocabularies developed at NCI.

Section 3 provides an introduction to the Cancer Data Standards Repository at NCI, and includes a review of the ISO/IEC 11179 standard on which it is based along with descriptions of the web interface and APIs to the caDSR.

1 CANCER BIOINFORMATICS INFRASTRUCTURE OBJECTS: CABIO

1.1 The caBIO Domain Objects and the Unified Modeling Language

The Unified Modeling Language (UML) is an international standard notation for specifying, visualizing, and documenting the artifacts of an object-oriented system. Defined by the Object Management Group (OMG), UML emerged as the result of several complementary systems of software notation, and has now become the de facto standard for visual modeling. In its entirety, the UML is composed of nine different types of diagrams. Each diagram type captures a different *view* of the system, emphasizing specific aspects of the design such as the class hierarchy, message-passing behaviors between objects, the configuration of physical components, and user interface capabilities. In this section we describe the UML class diagram as it is used to depict the caBIO class objects.

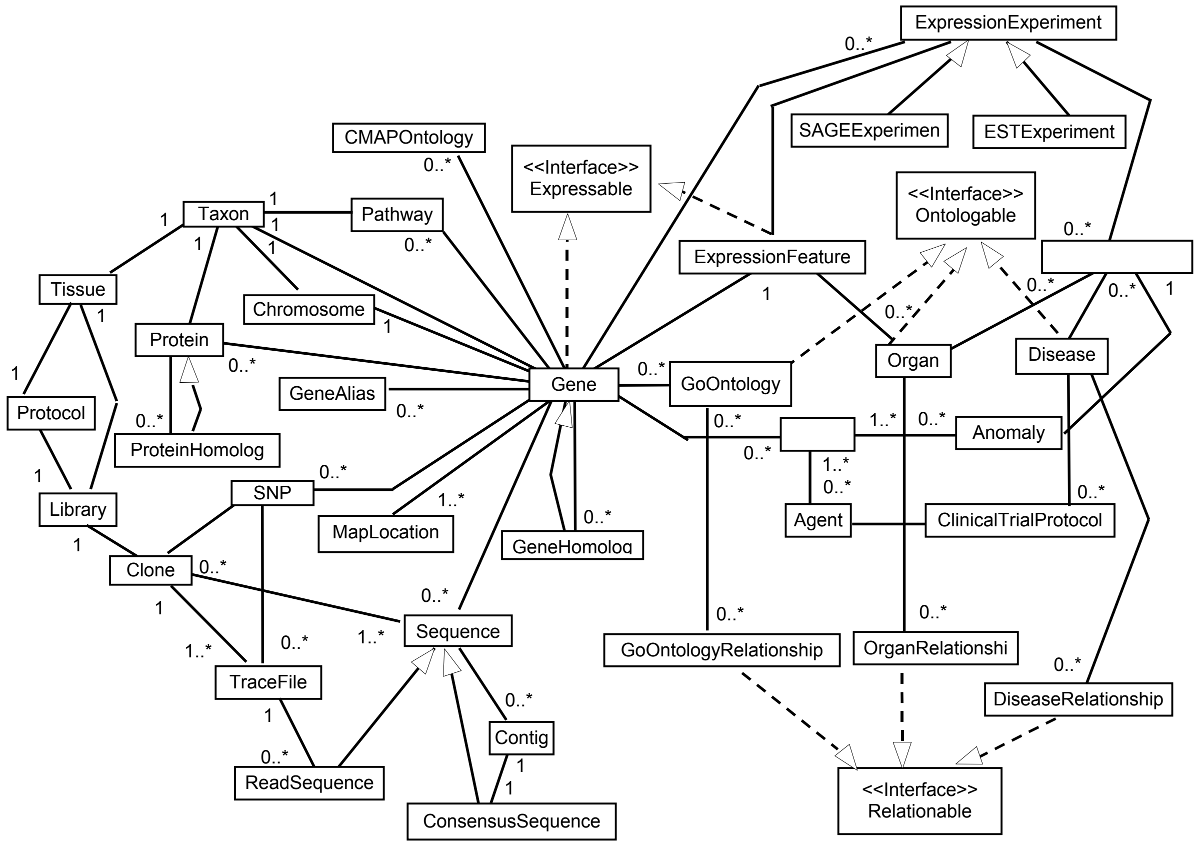


Figure 1.1.1. UML class diagram.

Figure 1.1.1 is not an exhaustive catalog of all objects in caBIO but, instead, depicts a subset of objects that are primary to bioinformatics applications and cancer research. Class objects can have a variety of possible relationships to one another, including "is derived from," "contains," "uses," "is associated with," etc. UML provides specific notations to designate these different kinds of relations, and enforces a uniform layout of the objects' attributes and methods — thus reducing the learning curve involved in interpreting new software specifications or learning how to navigate in a new programming environment.

Figure 1.1.2 (a) is a schematic for a UML class representation, and 1.1.2(b) is an example of how a simple class object might be represented in this scheme. The enclosing box is divided into three sections: The topmost section provides the name of the class, and is often used as identifier for the class. The middle section contains a list of attributes (data members) for the class. The bottom section lists the object's operations (methods). In the example below, (b) specifies the *Gene* class as having a single attribute called *sequence* and a single operation called *getSequence()*:

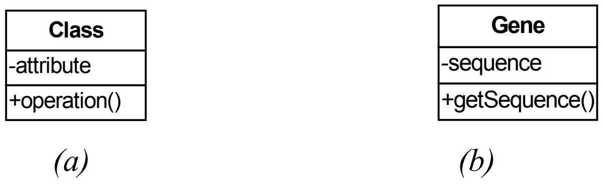


Figure 1.1.2. (a) Abstract schematic for a UML class. (b) A simple class called *Gene*.

The operations and attributes of an object are called its features. The features, along with the class name, constitute the signature, or classifier, of the object. The UML provides explicit notation for the permissions assigned to a feature. UML tools vary with how they represent their private, public, and protected notations for their class diagrams.

The caBIO development team uses Rational Rose™ software, a UML modeling tool available from Rational Software, Inc. Rational Rose uses variants on a lock and key icon (refer to Figure 1.1.3); simple tools use a "-" prefix for private features, a "+" precedes public features, and protected features have a "#". In the above example, the *Gene* object's *sequence* attribute is private and can only be accessed using the public *getSequence()* method.

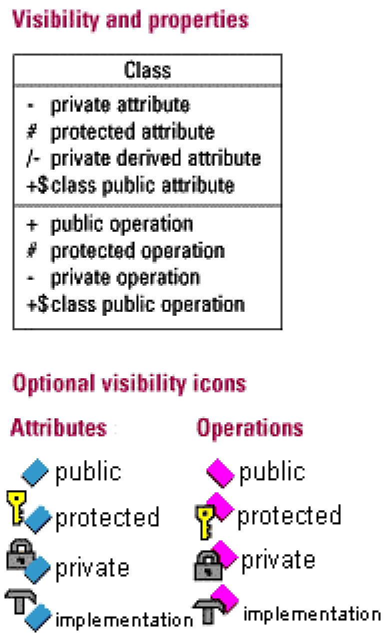


Figure 1.1.3. Rational Rose access modifier representations.

The caBIO classes represented in Figure 1.1.1 show only class names; both the operations and attributes are suppressed in that diagram. This is an example of a UML *view*: Details are hidden where they might obscure the bigger picture the diagram is intended to convey. Most UML design tools, such as Rational Rose, provide means for selectively suppressing visible details without removing the information from the underlying design model. In Figure 1.1.1, the emphasis is on the relationships that are defined among the objects, rather than on any particular class's features.

In more detailed class diagrams, it is common practice to display only those features that are part of the object's interface. An interface is the externally visible behavior of a class or component. In most cases, this means that only the object's public methods are shown, as the attributes are generally private or protected. The most salient information contained in Figure 1.1.1 is the objects' names and their relationships to one another, which are described next.

1.1.1 Relationships Among Classes

A quick glance at Figure 1.1.1 shows that most of the other classes are organized around the *Gene* and *Sequence* classes. These two classes are themselves related to each other, by the *has-a* relation. More generally, the relationships occurring among the caBIO objects are of three types: **association, aggregation, and generalization**. The most primitive of these relationships is association, which represents the ability of one instance to send a message to another instance. The relationship between the *Gene* and *Sequence* classes is an example of an association and is depicted by a simple straight line connecting the two classes.

Optionally, a UML relation can have a label providing additional semantic information, as well as numerical ranges such as 1..*n* at its endpoints. These cardinality constraints indicate that the relationship is one-to-one, one-to-many, many-to-one, or many-to-many, according to the ranges specified and their placement. For example, the *Gene-to-Chromosome* relation in Figure 1.1.1 is many-to-one.

UML relations may also have directionality, as in Figure 1.1.4. Here, a *Library* object is uniquely associated with a *Protocol* object, with an arrow denoting unidirectional navigability. Specifically, the *Library* object has access to the *Protocol* object (i.e., there is a *getProtocol()* method), but the *Protocol* object does not have access to the *Library* object.

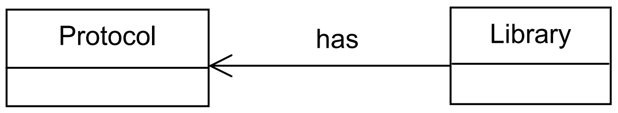


Figure 1.1.4. A one-to-one association with unidirectional navigability.

Figure 1.1.5 depicts a bidirectional many-to-one relation between *Sequence* objects and *Clone* objects. Each *Sequence* may have at most one *Clone* associated with it, while a *Clone* may be associated with many *Sequences*. To get information about a *Clone* from the *Sequence* object, we call the *getSequenceClone()* method. Each *Clone* in turn can return its array of associated *Sequence* objects using the *getSequences()* method. This bidirectional relationship is shown using a single undirected line between the two objects.

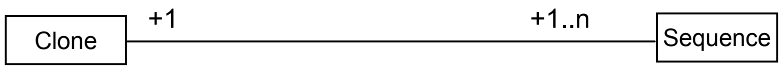


Figure 1.1.5. A bidirectional many-to-one relation.

The next relationship exhibited by caBIO objects is aggregation, which denotes a whole/part relationship. This relationship is exactly the same as an association with the exception that instances cannot have cyclic aggregation relationships (i.e., a part cannot contain its whole). So a *Library* can contain *Clones* but not vice-versa. Aggregation is represented by empty diamonds, as shown in the *Clone-to-Library* relation of Figure 1.1.6.

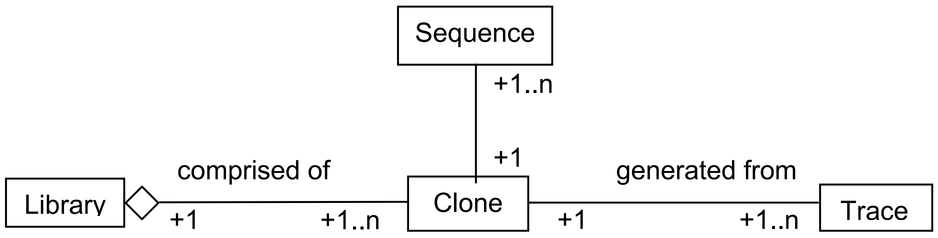


Figure 1.1.6. Aggregation and association.

Figure 1.1.6 shows a more complex network of relations. This diagram indicates that:

(a) one or more *Sequences* is contained in a *Clone*;
(b) the *Clone* is contained in a *Library*, which comprises one or more *Clones*;
(c) the *Clone* may have one or more Traces.

Only the relationship between the library and the clone is an aggregation. The others are simple associations.

In UML, the empty diamond of aggregation designates that the whole maintains a *reference* to its part. More specifically, this means that while the *Library* is composed of *Clones,* these contained objects may have been created prior to the *Library* object's creation, and so will not be automatically destroyed when the *Library* goes out of scope.

All information retrieval in caBIO is implemented by search methods associated with the caBIO objects. Indeed, the quintessential operation is to instantiate a "blank" instance of the desired object type, define appropriate search criteria to select specific instances of that type from the databases, and use the search results to populate either the original instance itself or, alternatively, an array of instances, in the event that more than one match is found. More information about this paradigm will be detailed in the next section, which provides a more in-depth overview of the caBIO domain objects. A comprehensive listing of all of the domain objects, their methods and their JAVADOC is available at the caBIO JavaDocs page.

The final relationship to be covered in this document is generalization**.** Figure 1.1.7 depicts a generalization relationship between the *SequenceVariant* parent class and the *Repeat* and *SNP* classes. Classes participating in generalization relationships form a hierarchy, as depicted here.

Generalization denotes a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element (it has all of its properties, members, and relationships) and may contain additional information. Both the *SNP* and *Repeat* objects follow that definition. The superclass-to-subclass relationship is designated by connecting unidirectional empty arrow heads, as shown in the *SequenceVariant-to-Repeat* and *SequenceVariant-to-SNP* relations of Figure 1.1.7.

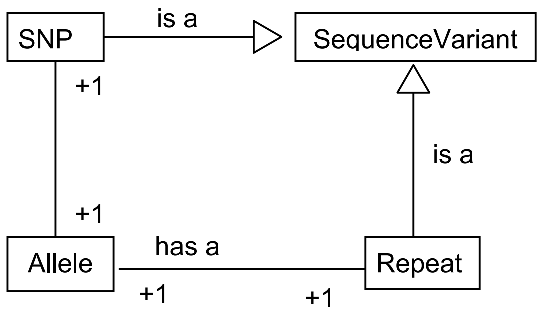


Figure 1.1.7. Generalization relationship.

As noted, the class diagram in Figure1.1.1 is a reduced form of the complete class diagram. A more complete view can be found at the caBIO Object Model pages. The contents of these pages were extracted automatically from the Java source code by the Rational Rose Web Publisher facility. Each page generates two frames; the tree structure on the left-hand side provides an index to the different views, and the right-hand side displays the diagrams associated with the currently selected view. To view the full caBIO class diagram:

1. Expand the *Logical View* folder located in the upper-left frame.
2. Double click on the *Main* icon.
3. Click on any object in the model to view its related attributes and associates.

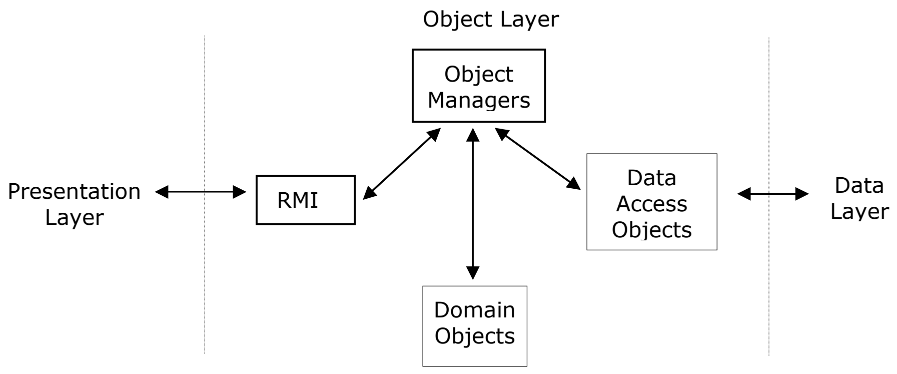


Figure 1.1.8. The caBIO object managers.

Double clicking on *Logical View/Managers* brings up a diagram illustrating the relationship between the various manager objects and the objects that they manage. As depicted schematically in Figure 1.1.8, the manager objects act as intermediaries to both the client side and the data sources on the back end, brokering requests and retrieving data as needed.

The *Logical View/ExpressionClasses* detail shows both inheritance relationships among expression classes as well as their usage of related classes.

Some of these diagrams introduce additional UML notation not yet discussed — specifically, the **interface** notation. For example, the *Logical View/Beans* diagram shows a few beans and a few interfaces. A UML diagram may explicitly label an interface using the <<interface>> notation, thus making it part of its stereotype name:

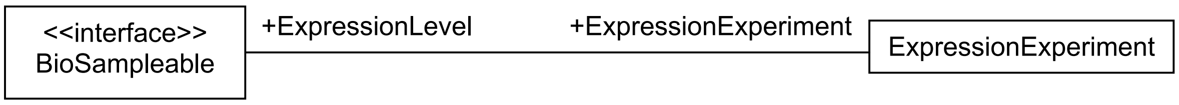
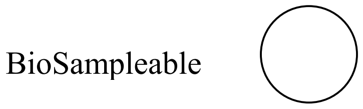


Figure 1.1.9. Interface as the stereotype name.

Alternatively, a diagram may use the so-called "lollipop notation," where an interface is represented by a small circle, as in:



The next two sections provide more details of the caBIO objects themselves. Section 1.2 describes the caBIO object hierarchy, their related classes, and the various search criteria objects. Section 1.3 discusses the design of the applications programming interfaces and the objects that implement them. Throughout these discussions, the user should refer to the caBIO Java Doc and caBIO Object Model web pages to locate further information about caBIO objects.

1.2 The Domain Object Hierarchy

As depicted in Figure 1.2.1, the domain objects in the *bean* package define a wide, shallow hierarchy. Here, we use the term "domain object" to refer to those Java bean classes that correspond to biological entities and bioinformatic concepts. All of the domain objects descend directly from *java.lang.object,* and only four of the domain objects have subclasses.

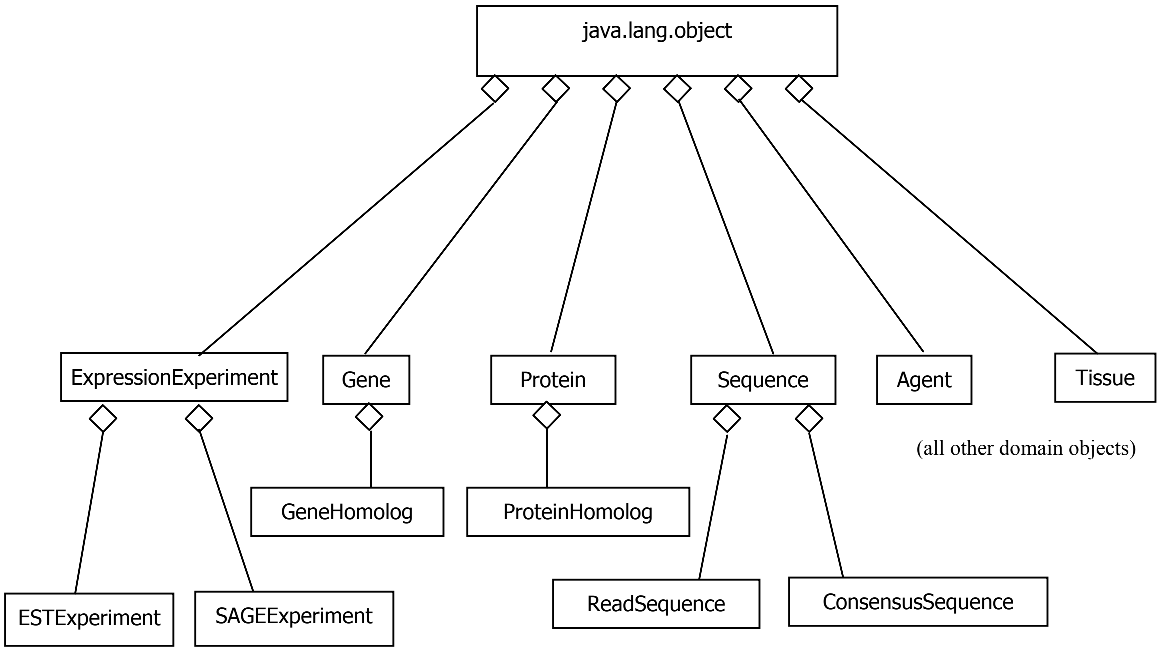


Figure 1.2.1. The domain object hierarchy.

Additional logical and tactical structure, however, derive from the interfaces implemented by some of these domain objects, and from related bean objects, such as the *SearchCriteria* beans and a small set of "relationable" objects.

Each domain object *Xxx* has a corresponding *SearchCriteria* named *XxxSearchCriteria.* Thus, for example, there is a *GeneSearchCriteria* object, a *LibrarySearchCriteria* object, etc., corresponding to the *Gene* and *Library* beans, respectively. The inheritance relations that hold among the *SearchCriteria* objects reflect their associations with the domain objects and, in addition, capture the interfaces that are implemented by some of the domain objects. Figure 1.2.2 shows the *SearchCriteria* object hierarchy.

All objects in the *bean* package implement the *java.io.serializable* interface. In addition, all domain objects implement *gov.nih.nci.caBIO.util.XMLInterface,* thus facilitating their transport to the Presentation Layer where the SOAP and HTTP applications programming interfaces (APIs) are implemented. Specifically, each domain object implements the following methods:

- *toXML()* – returns an XML-encoding of all of the object's "top-level" attributes (i.e., number and character-valued features), with all "deeper" information (e.g., arrays, embedded objects, etc.) encoded as XLinks. This is the default XML-encoding for the SOAP and HTTP interfaces.

The notion of an Xlink is similar to a pointer or reference in a programming language; the XML Linking Language (XLink) allows complex elements to be embedded in XML documents as URLs which can be subsequently deployed to retrieve the elements themselves.

- *toXMLDOC()* – returns an XML-encoding of *all* of the object's attributes; i.e., all XLinks are filled in one level deep. This method implements the *getHeavyXML* options used by the SOAP and HTTP interfaces.
- *toXMLProcessor(...)* – takes a list of *fillin* tags specifying which XLinks are to be selectively expanded in the XML-encoding.

Three additional interfaces are defined in *gov.nih.nci.caBIO.bean*: *Expressable, Ontologable,* and *Relationable.* Currently, only the *Gene* object implements *Expressable.* This interface defines a single method: *getExpression()*, which returns an array of *ExpressionExperiments* containing expression levels for the gene. But the definition of "expressable" as an interface also posits an implicit relationship between the *Gene* class and the *ExpressionExperiment* objects in the hierarchy, as each of these implements the method *getExpressables(),* which returns an array of *Expressable* objects.

A more complex set of relationships derives from the *Ontologable* and *Relationable* interfaces. The *Organ, Disease, CMAPOntology*, and *GoOntology* classes all implement the *Ontologable* interface, as each of these object types defines entities occurring in externally defined ontologies or taxonomies. The Gene Ontology Consortium, for example, defines three gene ontologies, based on molecular function, biological process, and cellular location of the gene. Similarly, the CMAP ontology maps genes according to functional classifications. Other controlled vocabularies, such as those defined by the Enterprise Vocabulary Services (EVS) at NCI, define disease and organ taxonomies.

Each of the domain objects implementing the *Ontologable* interface implements the *getChildRelationships()* and *getParentRelationships()* methods, which can be applied to get the parent and descendant terms in the vocabulary where the object's name is defined. The return type of both of these methods is actually an array of *Relationable[]* objects – i.e., objects that implement the *Relationable* interface. Objects in the bean package implementing the *Relationable* interface are the *OrganRelationship, DiseaseRelationship, CMAPOntologyRelationship,* and *GoOntologyRelationship* classes.

Each instance of a relationship object stores its relationship *type* (child/parent) and the set of *Ontologable* objects participating in that relationship. For example, an *Organ* representing the heart might have a parent relationship to two other *Organs* representing the left and right ventricles. The parent's method *getChildRelationships()* would return this (object-ified) relationship, and the relationship, in turn, would provide access to the *Ontologable* children stored there. More specifically, all *Relationable* objects implement the *getOntologies*() method for just this purpose.

While the inheritance hierarchy for the domain objects does not reveal these interface implementations, the hierarchy of *SearchCriteria* objects (Figure 1.2.2) does. As described in the general discussion of the caBIO APIs in the following section, the *SearchCriteria* objects are a critical part of the infrastructure that provides caBIO's powerful database search mechanisms.

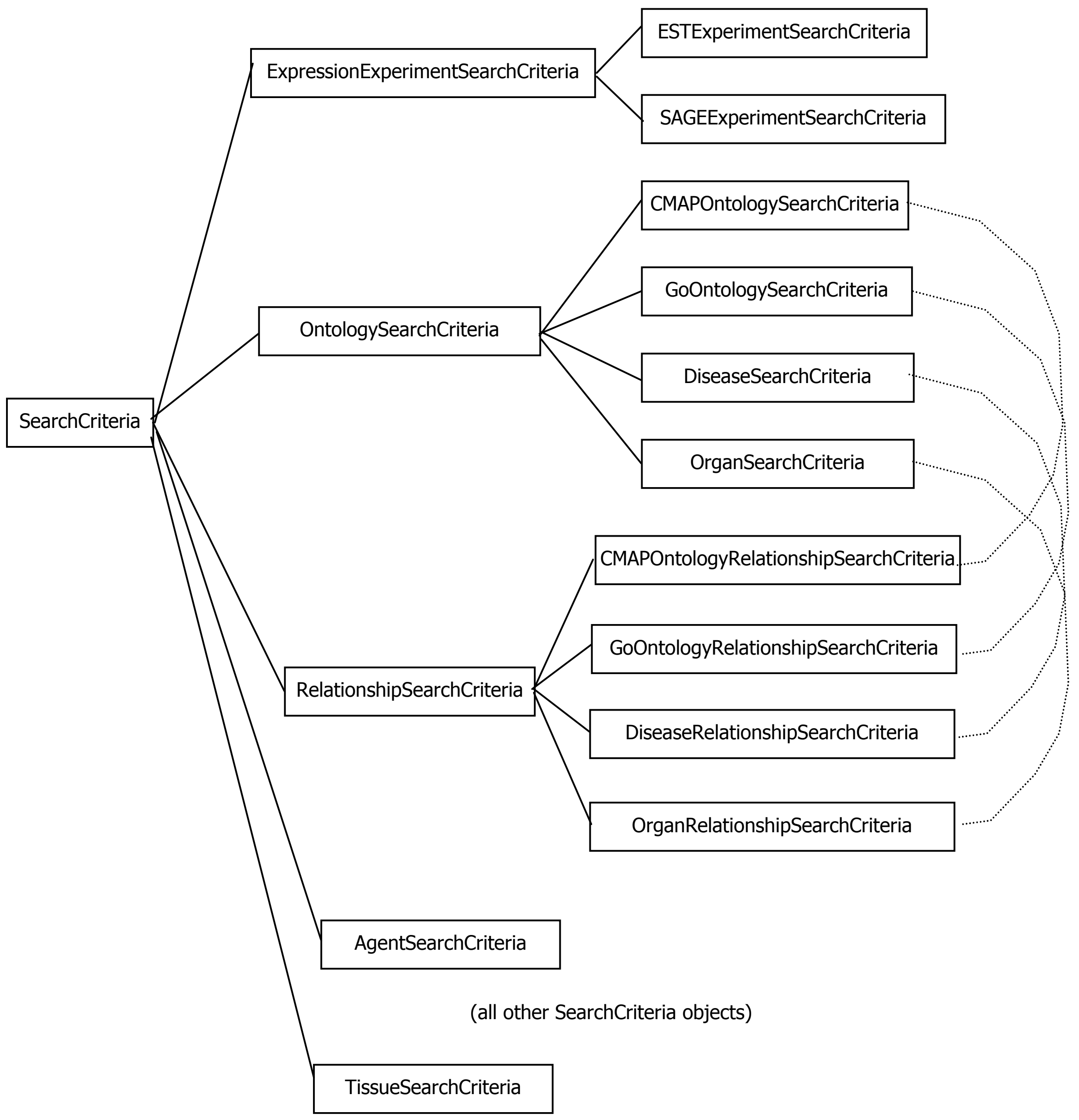

ExpressionExperimentSearchCriteria
ESTExperimentSearchCriteria
SAGEExperimentSearchCriteria
CMAPOntologySearchCriteria
GoOntologySearchCriteria
OntologySearchCriteria
DiseaseSearchCriteria
SearchCriteria
OrganSearchCriteria
CMAPOntologyRelationshipSearchCriteria
GoOntologyRelationshipSearchCriteria
RelationshipSearchCriteria
DiseaseRelationshipSearchCriteria
OrganRelationshipSearchCriteria
AgentSearchCriteria
(all other SearchCriteria objects)
TissueSearchCriteria

Figure 1.2.2. The *SearchCriteria* inheritance hierarchy.

Additional information about the caBIO objects in the *bean* package can be found in Section 1.3, and a quick reference guide is included in Section 1.9. In particular, the API descriptions in the next section offer important insight into the relationships between *SearchCriteria* and domain objects. The online JavaDocs pages provide comprehensive specifications of the objects, interfaces, and packages which together define the caBIO architecture and services.

1.3 caBIO API Overview

The caBIO infrastructure comprises an *n*-tiered architecture, as depicted in Figure 1.3.1. At the back-end are various caBIO databases, flat files, and URLs to external databases and public websites. At the front end is a Presentation Layer providing APIs capable of supporting a wide variety of programming languages. The focus of the first part of this section is on the Presentation Layer and its interfaces to the various clients as well as to the Java bean classes that implement the Object Layer. The second part of this section focuses in greater detail on the caBIO domain objects defined in the *bean* package. Throughout this discussion the reader may wish to refer to the caBIO JavaDocs pages for additional information.

At the heart of the caBIO architecture are the bean classes that comprise the Object Layer. Although the complete caBIO implementation includes a number of packages, this discussion is limited to the *bean, das, servlet, manager*, and *webservices* packages that together implement the APIs.

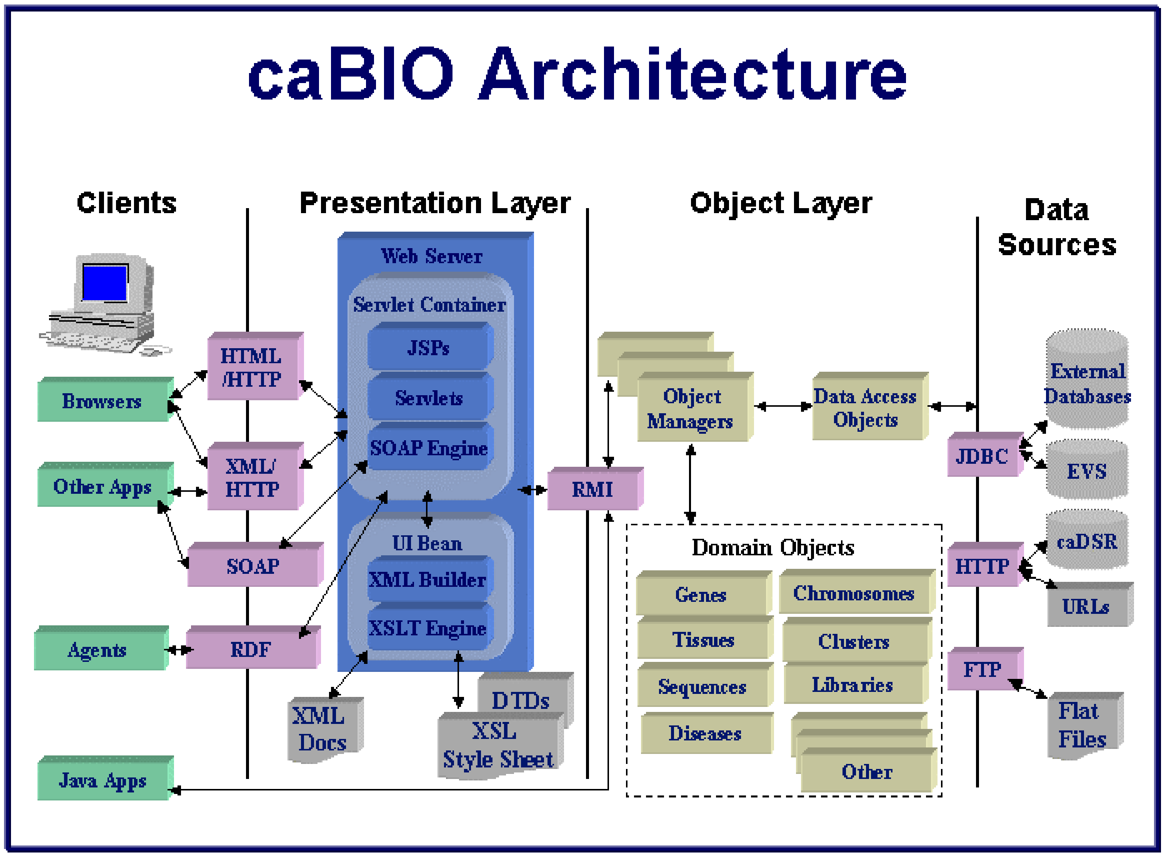


Figure 1.3.1. The caBIO architecture.

As depicted in Figure 1.3.1, the objects themselves fall into three primary groupings: (1) domain objects that represent bioinformatic entities such as genes, chromosomes, sequences, tissues, etc.; (2) managerial objects supporting the infrastructure, and (3) data access objects interfacing to the back-end data sources.

Java applications send requests directly to the Object Layer using RMI, effectively bypassing the Presentation Layer. All other applications require some type of interface, such as the SOAP Engine or the JSPs provided by the Presentation Layer. We consider first how the Java API operates, as these same processes must execute on the back-end of all other interfaces.

1.3.1 The Java API Search/Retrieve Paradigm

The driving force behind the caBIO design is a data-mining paradigm, and the basic operation is a database search and retrieval, using a newly instantiated domain object as the target to be populated with the data that are returned. The domain objects are contained in the *bean* package. In addition to the methods that are particular to the individual classes, all domain objects implement the *search()* method. Each domain object also has an associated *SearchCriteria* object, which is the sole argument to the domain object's *search()* method. For example, corresponding to the *Gene* domain object, there is a *GeneSearchCriteria* object, and the syntax of the *Gene's* search method is:

```
myGene.search(myGeneSearchCriteria)
```

Each search criteria object has attributes that are used to constrain the search and define the type of information that will be returned in the result set. In addition to attributes that are particular to their associated domain objects, the object-specific search criteria classes inherit methods and attributes from a generic *SearchCriteria* object. Table 1.3.1 lists some of the more important inherited methods.

Method	Description
setMaxRecordset()	Sets the maximum number of result objects to return (default 1000)
setOrderBy()	Sets the order by clause for the SQL
setReturnCount()	Specifies that the number of objects found should be included in the search result (default false).
setReturnObjects()	Specifies that the objects themselves should be included in the search result (default true)
setStartAt()	Sets the number of the first member of the result array

Table 1.3.1. Common methods implemented by all *SearchCriteria* objects.

The return value of a domain object's *search()* method is always an object of type *SearchResult*, whose attributes and methods approximately mirror the attributes and methods of the generic *SearchCriteria* object. For example, the *SearchResult* object has a method *getCount()* that returns the number of objects that matched the specified criteria. This method's return value is only defined, however, when the associated *SearchCriteria* object specified that the *number* of objects found should be included in the search result.

Similarly, the *SearchResult* object's method *getResultSet()* returns an array of objects only if the *SearchCriteria* specified that the objects themselves should be included in the search result. Setting this last option to *false* is useful in situations where the only information that is needed is the count of objects satisfying the criteria, and not the objects themselves. By default, *setReturnCount()* is *false* and *setReturnObjects()* is true, unless the *SearchCriteria* options are explicitly reset using these methods.

The *SearchResult* object's methods *getStartsAt()* and *getEndsAt()* return the array index of the first and last objects in the result array, respectively. While these methods might seem at first to be gratuitous, they are actually a critical part of the caBIO API design, which provides a "throttling" mechanism to limit the number of results returned on any single query. By default, the maximum number is 1,000, but, as indicated here, this can be reset to a smaller or larger number as desired.

As we will see, these same mechanisms are implemented in the subsequent API layers providing interfaces to other programming languages. Given the enormous amount of currently available bioinformatic data and their exponential rate of growth, this ability to receive large amounts of data in bursts is indispensable.

Several additional methods further facilitate situations where a very large set of objects match the search criteria. The *SearchResult* object's method *hasMore()* returns *true* when further results are available that are not included in the current *SearchResult*. In this case, the *SearchResult* object's method *getNextCriteria()* can be used to return a new *SearchCriteria* object whose starting index picks up where the previous result set left off.

Other methods provided by the generic *SearchCriteria* object provide means of determining whether or not a specific criteria has been defined, removing previously set criteria, and/or adding new criteria to the current collection. Object-specific attributes are settable for the *SearchCriteria* objects associated with specific domain objects. For example, the *GeneSearchCriteria* object includes methods for specifying the desired gene symbol, chromosome id, organism, etc.

SearchCriteria objects can also be embedded in one another to specify more complex queries, via the method *putSearchCriteria()*. For example, to extract the set of all pathways containing the gene PTEN, one can:

1. Create a *GeneSearchCriteria(), myGenesCriteria.*
2. Invoke: *myGenesCriteria.SetSymbol("*PTEN*").*
3. Create a *PathwaySearchCriteria(), myPathsCriteria.*
4. Invoke *myPathsCriteria.putSearchCriteria (myGenesCriteria).*
5. Create a *Pathway* domain object, *myPathway.*
6. Invoke *myPathway.search(myPathsCriteria).*

The result of step 6 will produce an array of pathways containing the PTEN gene. This example demonstrates another feature of the caBIO design: Each domain object also serves as a factory for creating multiple instances of that object. Reviewing the steps outlined above, we can generalize the factory process as follows:

1. Instantiate a *new* domain object of the desired type.

2. Create a *new SearchCriteria* for that object, and set its attributes.

3. Execute the domain object's *search()* method on that *SearchCriteria*, and store the results in a generic *SearchResult* object.

4. Invoke the *getResultSet()* method on the *SearchResult* object and typecast its return value to an array of the same type as the original domain object.

The installation of the caBIO software for a Java client (see Section 1.4) includes a Java archive file (*cabio.jar*) defining all of the caBIO domain objects as well as the protocols and server information required to issue RMI requests to the caBIO servers. The logistics of retrieving data for a Java client are as follows. First, the Java client application declares a new instance of the object type of interest. For example, the client executes the statement:

```
Gene myGene = New Gene();
```

This instantiation of the new *Gene* object alerts the *GeneManager* on the caBIO server, and causes a proxy for that manager to become resident on the client machine for the duration of the application. All RMI requests on the *Gene* object from that point forward will be handled by the protocols defined for the manager and proxy objects.

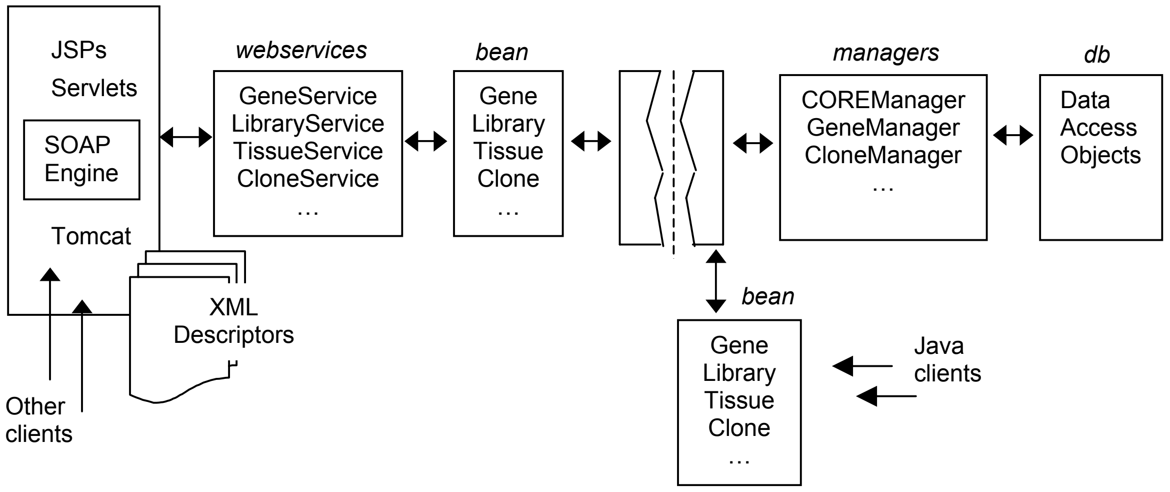


Figure 1.3.2. The logical deployment of the caBIO packages for data retrieval.

If a subsequent request such as *myGene.getSequences()* is issued, the instantiation of the resulting *Sequence* objects will in turn be handled by a remote *SequenceManager* and a local proxy. A logical view of the caBIO object managers is available in the caBIO Object Model pages, and their specifications can be viewed on the JavaDocs pages. By virtue of the caBIO object managers, the network details of the communication to the caBIO server are abstracted away from the Java developer. Figure 1.3.2 is an alternative representation of the caBIO architecture, emphasizing how some of Java packages in caBIO are deployed to implement the different APIs.

A general rule of thumb for all caBIO domain objects is that only those attributes represented by primitive data types (e.g., integer, string, float, etc.) are returned directly with a retrieved object. For more complex types such as objects and arrays, access to these entities is provided, but the objects themselves are not. Instead, the "containing" object provides methods for retrieving these embedded objects recursively. Thus for example, a *Gene* object provides a method called *getSequences()* to retrieve its associated genomic sequences once the *Gene* itself has been obtained.

Similar mechanisms for returning only top-level information — with the option of drilling down further where desired — are implemented in the SOAP and HTTP interfaces described below. The SOAP and HTTP interfaces also provide mechanisms for controlling the number of objects returned for a single query, corresponding to the *SearchCriteria* object's *setMaxRecordset()* and *setStartAt()* methods.

An important difference between the Java API and other interfaces to the caBIO is that only Java applications have direct access to the domain objects and their methods, as they are defined in a local jar file.

1.3.2 The SOAP API

caBIO's Simple Object Access Protocol (SOAP) API is provided for non-Java applications written in languages such as Perl, C, Python, etc. SOAP is a lightweight XML-based protocol for the exchange of information in a decentralized, distributed environment. It consists of an envelope that describes the message and a framework for message transport. caBIO uses the open source Apache SOAP package, in combination with Jakarta Tomcat, to provide its web services to users.

It is up to the application developer to select and install a SOAP client for the development environment. Section 1.5 includes an example application that uses the SOAP::Lite client for Perl. The SOAP request issued by the client is formatted as an XML document that is posted to the caBIO server at a listening port reserved for SOAP requests. The response returned by the server is also an XML document

All of the caBIO domain objects are "XML Aware" and are capable of serializing themselves to XML for transport to a wide variety of platforms. These domain objects, however, do not directly receive the SOAP requests, as they are first parsed by objects in the Presentation Layer. Specifically, caBIO's SOAP server receives requests from the remote SOAP client, and forwards these to an appropriate class in the *webservices* package.

Each SOAP web service defined in the *webservices* package has methods mirroring those defined for a corresponding domain object in the *bean* package. For example, the SOAP *GoOntologyService* has methods called *getChildRelationships(), getParentRelationships()*, *getHomoSapienGenes(),* and *getMouseGenes(),* corresponding to the *GoOntology* domain object's methods of the same name.

All of the SOAP services inherit from the parent class *WebCriteriaInterpreter*, whose two methods are *readInCriteria()* and *searchObjects().* Each of these methods takes two arguments: a hashtable of tag/value pairs and a string specifying the object type to search for. The return value of *readInCriteria()* is a *SearchCriteria* object; the return value of *searchObjects()* is a *SearchResult* object. Thus, the SOAP service classes in the *webservices* package can use the parent class's methods, in combination with their own method specializations, to transform the incoming HTTP requests to equivalent Java method invocations that can be passed on to domain objects in the *bean* package.

Once the results of the search have been obtained from the back-end data sources, the domain objects' *toXML()* methods are applied to return XML-encoded responses to the SOAP service classes in the *webservices* package. There, the service classes can forward these responses to the SOAP server, which, in turn, forwards these as strings to the SOAP client.

The SOAP API provides a throttling mechanism that is similar to that described above for the Java API. In the SOAP API, this is achieved by using the XML Linking Language (XLink), which effectively provides a way of embedding "pointers" in the XML output, thus reducing the amount of information returned. As with the Java API, all attributes represented by simple data types (i.e., numbers and strings) are included directly in the SOAP output. More complex data types, such as structured objects and arrays, are returned as Xlinks providing URLs where the data can be fetched recursively.

Two additional parameters can be included in the SOAP request, however, which specify that the XML-encoded response should also expand either all or only selected Xlinks. These additional parameters are *returnHeavyXML,* which takes the values 0/1, and *fillInObjects*, which takes a list of comma-separated arguments specifying the Xlink tags that should be expanded. The *returnHeavyXML,* when set to true, opens up all of the embedded Xlinks one level deep. Detailed illustrations on how to use these parameters are provided in the SOAP API examples section (Section 1.5).

1.3.3 The HTTP Interface

The Hypertext Transfer Protocol (HTTP) provides a non-programming interface for accessing caBIO data. Using the HTTP interface does not require any additional software other than a web browser such as Internet Explorer or Netscape Communicator. Like the SOAP API, the HTTP interface uses the domain objects in the Object Layer of the caBIO *n*-tier model to communicate with the back-end data sources. The HTTP interface forwards its requests as URLs to a Java servlet running in the Presentation Layer, called *getXML*. The *getXML* servlet is defined in caBIO's *servlet* package, and has methods that receive requests from HTTP clients, forward messages to the respective domain objects for processing, and return the results as XML-encoded responses to the HTTP clients.

The HTTP request parameters correspond to methods in the respective *SearchCriteria* object associated with the domain object being queried. One can narrow down a search by using as many parameters as required. The XML output returned from an HTTP request is similar to the XML output from a SOAP client request. The XML output received by the HTTP client also embeds XLinks to limit the amount of information returned in a single response.

As with the SOAP API, the HTTP interface allows the user to further expand these XLinks by using *returnHeavyXML* and *fillInObjects* as additional parameters in the HTTP request. The number of top-level objects returned can also be controlled by specifying values for *StartAt* and *EndAt* in the HTTP request or, alternatively, by using the *ResultCount* parameter. Details and examples of using the HTTP interface are given in the HTTP interface discussion in Section 1.6.

1.4 The caBIO Java API

The caBIO Java API provides an enhanced object-oriented development environment for bioinformatics researchers, along with access to customized data sources for plumbing the molecular basis of cancer and manipulating clinical data. The data sources include many of the NCICB databases specially curated for cancer research, as well as many of the public databases at NCBI, EMBL, and others. These data sources are described in more detail in the caBIO Data Sources section (Section 1.7). Table 1.4.1 lists some of the Java packages contained in the API, along with a brief description of their functions.

Package name	Description
gov.nih.nci.caBIO.bean	Object representations of bioinformatic concepts and entities, such as *Gene, Chromosome, Library,* etc.
gov.nih.nci.caBIO.db	Data access objects interfacing to the data sources
gov.nih.nci.caBIO.evs	Object representations of EVS concepts and entities, including *Concept, Metaphrase,* and *SemanticType*.
gov.nih.nci.caBIO.manager	Object manager classes to support RMI on the server.
gov.nih.nci.caBIO.net	Proxy manager classes to support RMI on the client – these objects encapsulate the network protocols used by the domain objects to communicate with the server, and thus abstract RMI implementations away from the user.
gov.nih.nci.caBIO.servlet	Classes supporting JSP pages in the Presentation Layer; encapsulates the caBIO servlet implementations.
gov.nih.nci.caBIO.util	Classes supporting XML-encoding and other types of serialization; includes the DOM parser and Scalable Vector Graphics (SVG) pathway wrappers.
gov.nih.nci.caBIO.util.das	Classes providing access to the Distributed Annotation Server at UCSC. The classes in this package were auto-generated from the DAS DTD's using the Sun JAXB "xjc" tool.
gov.nih.nci.caBIO.webservices	Classes implementing the SOAP services in the Presentation Layer.

Table 1.4.1. Packages included in the caBIO Java API.

The caBIO objects in the *bean* package — also referred to as the caBIO domain objects simulate the behavior and relationships of actual bioinformatic components such as genes, chromosomes, sequences, libraries, clones, ontologies, etc. They provide access to a variety of bioinformatic data sources including Unigene, LocusLink, Homologene, GoldenPath, and NCICB's CGAP (Cancer Genome Anatomy Project) data repositories. Hence, a gene can get its ESTs, SNPs, or clones; an SNP can provide access to the TraceFiles that were used to identify it; and a chromosome can report the taxon in which it is defined.

Java applications can access the caBIO data sources directly through these domain objects; the network details of communication to the data servers are abstracted away from the developer by the supporting packages in Table 1.4.1. Thus, the developers need not deal with issues such as RMI and can instead concentrate on the biological problems at hand.

The implementation of a separate data layer allows the domain objects to act independently of the actual data storage facilities. This allows the data layer to migrate as necessary to increase performance or provide new data stores, without impact to the application programs. In particular, the Data Access Objects (DAOs) in the *db* package enable platform independent persistence of these domain objects, and the relational mappings provided by the DAOs are optimized for the data queries presented by the domain objects.

Another important feature of the caBIO data access objects is their ability to cache and manage large amounts of data. Coupled with the throttling mechanisms deployed to control the flow of data through the system, this design provides optimal response time to all users of the system.

The caBIO domain object classes are what most developers will use to access the information available from the caBIO servers. These domain objects are available as Java beans in the caBIO jar file that is downloaded with the caBIO installation. In most cases, the developer need not look beyond the caBIO *bean* package to accomplish his or her goals.

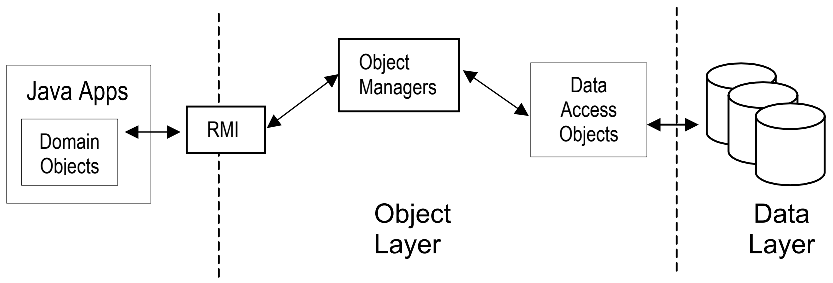


Figure 1.4.1. The caBIO Java API

More ambitious applications may require adapting and extending the basic caBIO platform, and/or installing the data sources as a local resource. The entire caBIO source code is available for download.

As an example of the types of applications that can be built using these development tools, visit the CMAP website, which is implemented using the caBIO objects described in this manual.

1.4.1 Installing the caBIO Java API

The caBIO Java API requires JDK 1.3.1 or higher. If JDK is not already installed in your system, follow the instructions from the Java installation and tutorial website for details on installing JDK.

The caBIO Java API can be downloaded from the caBIO technical resources website. After filling in your user name, institution, and email address, you are given the option of downloading either the caBIO binary jar file directly or the zipped caBIO_Demo. This discussion assumes you have downloaded the Demo package.

Unzip these files into a working directory of your choice; for the purposes of this discussion we will assume you are using c:\caBIO. Examine the file structure in your newly created directory. In addition to the top-level files, you will find several subdirectories. The directory named *jars* contains the caBIO Java archive, *caBIO.jar*, along with the other following jar files:

- xerces.jar (the Apache Xerces-J XML parser)
- jaxp-api.jar (the Sun API for XML processing)
- jaxb-rt-1.0-ea.jar (the Sun architecture for XML binding)
- crimson.jar (the Apache Crimson Java XML parser)
- soap.jar (Apache SOAP)

1.4.2 Defining the ClassPath

In order to compile and/or execute caBIO applications, the Java compiler and runtime environments must be able to locate the caBIO class definitions as well as those for the classes in the additional jar files. This can be accomplished in three ways: (1) you can use a compile tool such as *ant*; (2) you can use the scripts provided with the caBIO_Demo download (*.bat* for Windows and *.sh* for Unix); or (3) you can set the `CLASSPATH` environment variable directly.

The first two methods are preferable, as hardcoding the locations of java archive files in your environment can create problems with versioning. Instructions for using the *ant* utility are included in the *readme* file that accompanies the distribution.

The two script files contained in the caBIO_Demo distribution are *compile_caBIO.** and *run_caBIO.** The compilation script explicitly specifies the classpath as an argument to the *javac* compiler. The execution script also specifies the classpath, and in addition, specifies the java.security file using the –D define flag. If you plan to use either of these batch files, ensure that the classpaths they specify concur with your installation of the corresponding jar files.

To set the `CLASSPATH` environment variable directly, Windows 98 users should modify the `autoexec.bat` file by adding the following (single) line:

```
CLASSPATH=%CLASSPATH%;c:\cabio\jars\xerces.jar;c:\cabio\jars\caBIO.jar;c:\cabio\jars\c
rimson.jar;c:\cabio\jars\soap.jar;c:\cabio\jars\jaxp.jar;c:\cabio\jars\jaxb-rt-1.0-
ea.jar;.
```

Note that this works only if you have previously defined the `CLASSPATH`; if not, you must use:
`Set CLASSPATH=c:\cabio\jars\... instead of CLASSPATH=%CLASSPATH%;c:\cabio\jars\...`

Users of Windows NT and Windows 2000 can enter this information directly, by clicking on *My Computer* → *Properties*, and selecting the *Advanced* tab, which brings up a dialog for editing your environment variables.

1.4.3 Compiling and Running the GeneDemo Program

The *javaDemos* directory contains both the source code as well as the executable class for a Java program called GeneDemo.java. Using the *.bat* file on Windows machines, you can now compile the demo by typing:

```
compile_caBIO.bat GeneDemo.java
```

at the command line in a DOS shell. Alternatively, if you have defined the classpath environment variable, you can just use

```
javac GeneDemo.java
```

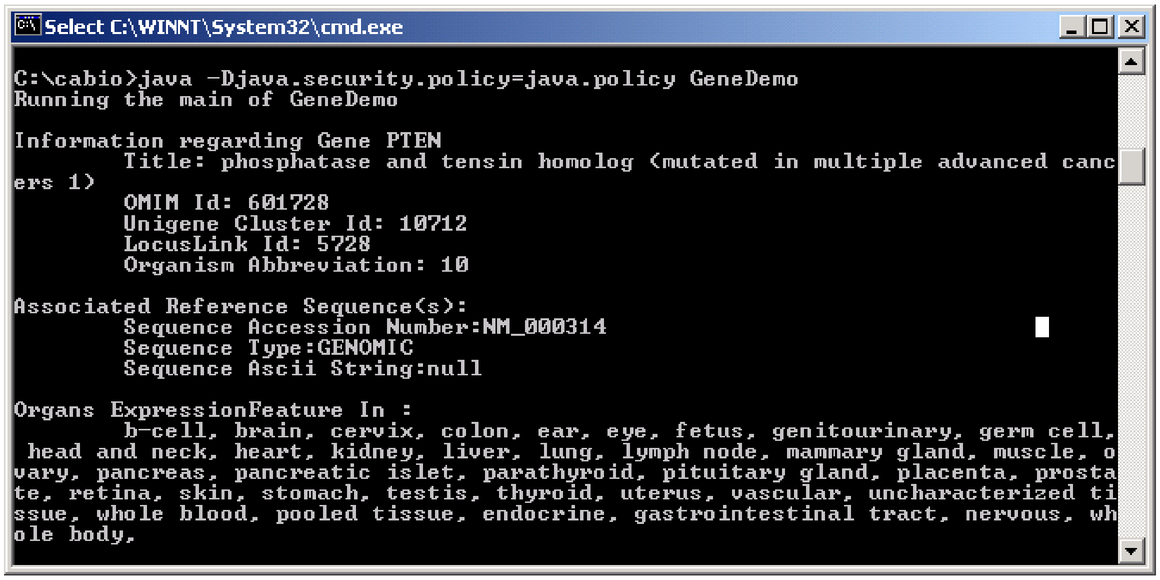
This will reproduce the java class file named GeneDemo.class, which you can then execute by typing:

```
java -Djava.security.policy=java.policy GeneDemo
```

The –D flag defining the security policy is provided to the RMISecurityManager class, which requires that you specify a security policy at runtime. The policies defined in the *java.policy* file protect your system — not the caBIO server — and you are free to edit these as you see fit. For example, a policy file granting full access permissions to everyone would contain the text:

```
grant {
  permission java.security.AllPermission;
};
```

By default, the policy file that you downloaded grants all permissions. The commented out section is an example of alternative settings you may wish to use. The following screen shot captures the first page of output that results from executing the *GeneDemo.class* file:



```
C:\cabio>java -Djava.security.policy=java.policy GeneDemo
Running the main of GeneDemo

Information regarding Gene PTEN
        Title: phosphatase and tensin homolog (mutated in multiple advanced canc
ers 1)
        OMIM Id: 601728
        Unigene Cluster Id: 10712
        LocusLink Id: 5728
        Organism Abbreviation: 10

Associated Reference Sequence(s):
        Sequence Accession Number:NM_000314
        Sequence Type:GENOMIC
        Sequence Ascii String:null

Organs ExpressionFeature In :
        b-cell, brain, cervix, colon, ear, eye, fetus, genitourinary, germ cell,
 head and neck, heart, kidney, liver, lung, lymph node, mammary gland, muscle, o
vary, pancreas, pancreatic islet, parathyroid, pituitary gland, placenta, prosta
te, retina, skin, stomach, testis, thyroid, uterus, vascular, uncharacterized ti
ssue, whole blood, pooled tissue, endocrine, gastrointestinal tract, nervous, wh
ole body,
```

1.4.4 Troubleshooting

The first place to look for errors is in the `CLASSPATH` definition; verify that all of the required jar files are present and under the correct subdirectories. It is also possible that you have installed everything correctly but that the executable has hit a firewall in trying to access the caBIO data services. For example, if you see the first line of output:

```
Running the main of GeneDemo
```

followed by the error message:

```
Proxy unable to contact Gene manager! Connection refused to host:...
```

then you have hit a firewall on your system and need to ask your system administrator to open the ports that the caBIO data services are using. Open another shell window and run the `netstat` system utility (while simultaneously running *GeneDemo*) to identify these ports.

A second problem, which can produce similar error messages like:

```
Proxy unable to contact Core manager! Connection refused to host:...,
```

can arise if you have:

(1) redefined the *java.policy* file, and/or

(2) installed the cabio.jar file in some location other than the default configuration.

In this case, the program will get past the Gene manager and produce the first few lines of output before running into trouble. To test this, redefine the *java.policy* file to grant all permissions (as outlined in the previous subsection), and rerun the program. If this solves the problem, then you will have established that the Core manager class (defined in caBIO.jar) was not able to locate the *java.policy* file. If you are not comfortable with using the open security policy, then you will need to reconfigure your setup so that the policy file and cabio.jar file have the following organization:

```
..\caBIO\
    java.policy
    jars\
      cabio.jar
```

1.4.5 Understanding the GeneDemo Program

The most important method — indeed the paradigmatic operation — on a caBIO object is the *search* method. Corresponding to each domain object is a *searchCriteria* object that can be deployed to retrieve objects of that type which satisfy user-specified criteria. For example, to obtain information about a particular gene we would:

1. Instantiate a new Gene object (e.g., *myGene*).
2. Instantiate a new *GeneSearchCriteria* object (e.g., c*riteria*).
3. Set the attributes of the *GeneSearchCriteria* to limit the search.
4. Call the Gene's search method with the *GeneSearchCriteria* object as the argument.

This approach is used in *GeneDemo.java* to retrieve all instances of *Gene* objects whose symbols match the string "PTEN." Specifically, the *setSymbol()* method of the criteria object is first applied, and a subsequent call to *myGene.search(criteria)* is then executed. Note that the return result needs to be typecast to *Gene[]*, as the *SearchResult* object's method *getResultSet()* returns a generic container.

The *Gene* object in this example and, more generally, every caBIO domain object, should be viewed as a "factory" that enables us to procure a collection of caBIO objects of that same type. Reviewing the steps outlined above, we can generalize the manufacturing process as follows:

1. Instantiate a *new* domain object of the desired type.
2. Instantiate a *new SearchCriteria* to be associated with that domain object, and set the attributes of that criteria object so as to limit the search.
3. Execute the domain object's *search()* method on that *SearchCriteria*, and store the results in a generic *SearchResult* object.
4. Invoke the *getResultSet()* method on the *SearchResult* object and typecast its return value to an array of the same type as the original domain object.

In this example, the search finds an array of genes, and the code then explores the features of each one in turn. Simple features whose values are just strings or numbers are printed directly to the screen. These include attributes like the gene's name, title, OMIM id, Unigene cluster id, LocusLink id, and organism abbreviation. More complex features represent embedded objects or arrays of simple elements or objects, and must be explored recursively.

For example, the gene's *getReferenceSequences()* method returns an array of sequences, and the features of each *Sequence* object are explored in turn. Similarly, the gene's *getDbCrossRefs()* method returns a hashtable or associative array of key/value pairs. The keys are stored in a Java enumeration variable and used to access successive element values. In contrast, the gene's *getExpressionFeature()* method does not return an array, but an *ExpressionFeature* object. In this case, the object's *getExpressedInOrgans()* method produces an array of string values that is iterated over.

The previous section described the caBIO architecture and the underlying design that drives the logistics of the *search()* methods and interactions between the domain objects, object managers, *SearchCriteria*, and *SearchResult* objects. The GeneDemo.java program demonstrates how these objects and devices can be deployed to extract information from the caBIO data sources.

1.5 The caBIO SOAP API

1.5.1 Introduction to SOAP

The Simple Object Access Protocol (SOAP) is a bridging technology that allows heterogeneous peers on diverse platforms to exchange structured data over the Internet via XML and HTTP. The caBIO project provides a SOAP interface for non-Java applications. In this model the client issues XML-encoded requests specifying the desired data services to the appropriate host address and port, and in exchange, receives XML-encoded responses.

The SOAP engine operates on three types of specification:

- The SOAP message's *envelope* specifications, which define the content type, intended recipient of the message, and whether it is optional or mandatory;
- The encoding rules, which specify the serialization method to be used in the exchange of application-specific data; and
- The RPC, which defines the conventions used in remote procedure calls and responses.

The simple example of a SOAP message below requests the price of apples from a host, www.foodprices.com:

```
POST /InPrices HTTP/1.1
Host: www.dictionary.com
Content-Type: application/soap; charset=utf-8
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body xmlns:m="http://www.dictionary.com " />
      <m:GetDefinition>
         <m:Item>Aperture</m:Item>
      </m:GetDefinition>
   </soap:Body>
</soap:Envelope>
```

In this example, the SOAP envelope specifications include only the content type (`application/soap`) and the recipient (`www.dictionary.com`). There is no mention of whether the message is mandatory or optional. The encoding rules are defined as `"http://www.w3.org/2001/12/soap-encoding"`. The RPC call specification is defined in the <`soap:Body` > element, where it requests the price of apples from the server.

1.5.2 The SOAP API and caBIO

As depicted in Figure 1.5.1, all of the caBIO data sources — including the internal databases as well as other NCI resources and external websites — are at the back-end of the caBIO infrastructure. The Object Layer consists of a set of object managers, data access objects, and a collection of classes representing biological and bioinformatic entities. All of the objects in this layer are implemented as Java bean classes and, as such, can be accessed by a Java program using remote method invocation (RMI).

The Presentation Layer provides a more generic interface to these same data for non-Java applications written in languages such as Perl, C/C++, Python, etc. caBIO uses the open source

Apache SOAP package, in combination with appropriate serialization methods for the Java beans, to achieve an application-independent interface. As described in the preceding general discussion of the caBIO API's (Section 1.3), all of the caBIO objects are "XML Aware" and are capable of serializing themselves to XML for transport.

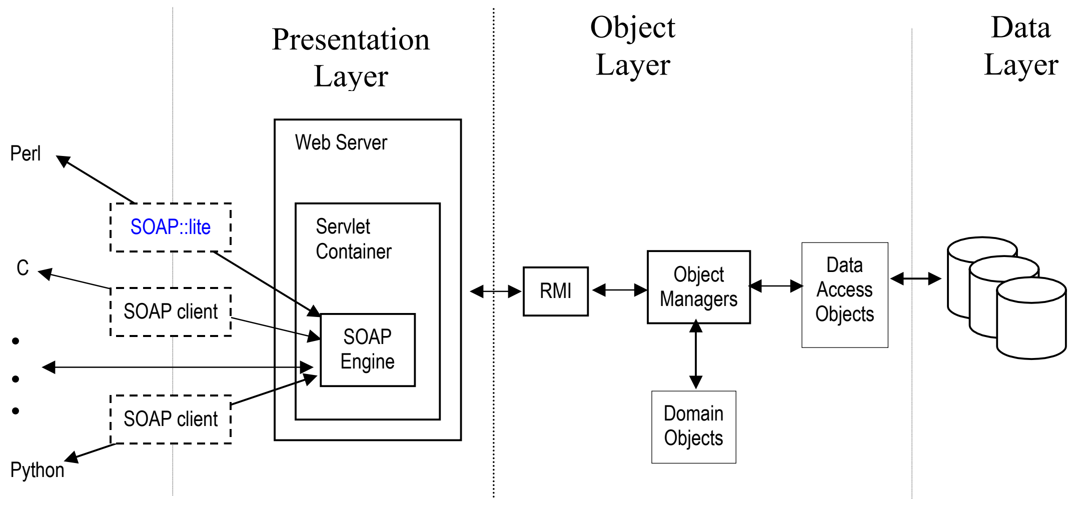


Figure 1.5.1. The caBIO architecture and the SOAP interface.

caBIO's SOAP API can be used as an interface to *any* language-specific application. In theory, the remote application could in fact communicate directly with the SOAP server without any additional layers of interfacing. In practice, however, this would involve a good deal of effort, as it requires explicitly wrapping each request in a SOAP envelope, parsing the return message types, and network programming to establish and maintain reliable connections.

Most developers instead prefer to install a SOAP client package to handle the implementation of the envelope and the resolution of the SOAP types. A number of SOAP packages catering to different programming languages are available at Soapware and at http://www.w3.org/TR/SOAP/.

One such package is SOAP::LITE for Perl, which can be freely downloaded from the ActiveState website. The PERL example discussed below utilizes SOAP::Lite, which implements both a client- and a server-side SOAP implementation. Win 32 machines can download the *.exe* file from ActiveState and simply follow the accompanying installation instructions. The installer will automatically add the <perl soap:lite installation-directory>/bin to the system PATH.

1.5.3 Using the SOAP API with Perl and SOAP::LITE

1.5.3.1 Accessing the caBIO SOAP Services

The first thing you will need to determine before connecting to a SOAP server is the set of callable services it provides. To see the list of SOAP services provided by the NCICB server, point your browser to:

`http://cabio.nci.nih.gov/soap/services/index.html`

The links displayed on the deployed services page are known as the Uniform Resource Name (URN) identifiers for the information resources. Additional information about a deployed service can be obtained by clicking on the URN for that service. This will open a *Deployed Service Information* page, providing details on its properties. For example, clicking on the *urn:nci-gene-service* link displays the *ID, Provider Type, Provider Class,* and *Methods* properties.

The most important properties are *ID* and *Methods*. *ID* is the Uniform resource Identifier (URI) for the SOAP service; *Methods* enumerates the methods available from the service. For example, some of the methods provided by *GeneService* are *getGenes, getTaxons, getClones, getSequences,* and *getPathways*.

The caBIO SOAP services are implemented by the classes defined in the caBIO *webservices* package. You can get the details on these classes from the JavaDoc pages for that package. The caBIO architecture includes about thirty "service" classes, which implement communication between the caBIO domain objects and SOAP client applications via XML documents. Table 1.5.1 lists several of the most frequently used services and the java bean domain objects they provide access to. The example Perl application that follows demonstrates how information about a specific *Gene* object can be selectively extracted using SOAP::Lite.

SOAP service name	caBIO bean class
GeneService	*gov.nih.nci.caBIO.bean.Gene*
LibraryService	*gov.nih.nci.caBIO.bean.Library*
TargetService	*gov.nih.nci.caBIO.bean.Target*
AgentService	*gov.nih.nci.caBIO.bean.Agent*
PathwayService	*gov.nih.nci.caBIO.bean.Pathway*
ClinicalTrialProtocolService	*gov.nih.nci.caBIO.bean.ClinicalTrialProtocol*
GenericObjectService	*gov.nih.nci.caBIO.bean*

Table 1.5.1 Frequently used caBIO SOAP services.

1.5.3.2 Accessing the GeneService using SOAP::Lite

SOAP::Lite is a collection of Perl modules that provide a simple interface to both the client and the server. Each SOAP::Lite method can be used for both setting and retrieving values. In the absence of any arguments, the current value is retrieved. When parameters are provided, the new value specified in the arguments will be assigned to the object referred to in the method. In the example, that follows, we will retrieve information about a specific gene, using the gene's symbol (PTEN) to select it.

Three mandatory arguments for accessing any SOAP service are:

- *server* – <the NCICB server URL>
- *port* – < the NCICB server listening port> e.g., 8080, 5080
- *method* – The requested SOAP service, e.g., *GeneService*

These arguments can be specified programmatically inside the Perl script, or at runtime on the command line. For example, given a Perl script called *geneClient.pl,* we can invoke it as follows:

```
geneClient.pl cabio.nci.nih.gov 5080 getGenes -symbol PTEN
```

Here, *cabio.nci.nih.gov* is the IP address of the SOAP server, 5080 is the listening port, and *getGenes* is the method of the *GeneService* we wish to access. The last argument is a specific parameter to *getGenes*, specifying that we would like to retrieve a *Gene* object whose symbol matches the string PTEN. Alternatively, we might specify these parameters in the Perl script itself, as:

```
$server = "cabio.nci.nih.gov";        # set the server variable
$port = "5080";                       # set the port variable
$method = "getGenes";                 # set the method variable
my %searchRec=();                     # declare a hashtable to store the search options
$searchRec{"symbol"} = "pTEN";        # initialize the symbol field in searchRec
```

Using either approach, our Perl script will still need to include additional variables specifying the URI for the SOAP service we wish to access, and a proxy path for message routing:

```
$URI='urn:nci-gene-service';            # set the URI variable to GeneService
$PROXY_PATH='/soap/servlet/rpcrouter'; # set the PROXY_PATH to (RPC|Message) Router
```

The proxy path specifies the endpoint service address and loads the required module. It is required for dispatching SOAP calls. `SOAP::Lite` provides explicit functions for these specifications: *uri()*and *proxy()*. Given the above $variable definitions, we can apply these as:

```
$s = SOAP::Lite;                                # declare a SOAP::Lite variable
-> uri($URI);                                   # set the SOAP::Lite uri()
-> proxy("http://$server:$port$PROXY_PATH");    # set the SOAP::Lite proxy()
```

1.5.3.3 Issuing a SOAP::Lite Service Request

Thus far, we have set up everything we need for the connection: the IP address for the SOAP server, the listening port for results, the URI for the SOAP service we wish to access, and a proxy path for message routing. We have also created an internal hashtable to store <tag/value> pairs for the search fields we will use; we have stored the pair <"symbol," "PTEN"> in that table, and we have assigned the string *getGenes* to the *method* variable. All that is left now is to declare a variable to store results in, and a way of actually invoking the desired method. We can do this as follows:

```
$som=$s->$method(SOAP::Data->type(map => \%searchRec));
```

In general *SOAP::Data* is used to specify a value, a name, a type, a URI, or attributes for SOAP elements. In this example we have used it to specify that the argument, *searchRec*, is a ***map*** type, since it is essentially a two-dimensional array. Alternatively, we might have specified `value(), name(), uri(),` or `attr()` in place of *type()*.

The return object *$som* is a SOAP::SOM object and can be used to access the returned values. If a fault element is in the message, *$som->fault* will be defined. Additional information, including *faultdetail, faultcode,* and *faultstring*, is also available from the *$som* object. If the request was successful, the response XML can be retrieved and saved by calling *$som->result* as follows:

```
$xmldoc = $som->result;            # get the result
open (OUT, ">pTEN.XML");           # open a file for output
print OUT $xmldoc;                 # write output to the file
print $xmldoc;                     # write to standard output
```

1.5.3.4 The complete *geneClient.pl* Perl Script:

```
use SOAP::Lite;
use HTML::Entities;
$URI='urn:nci-gene-service';
$PROXY_PATH='/soap/servlet/rpcrouter';
my %searchRec=();
   $server = "cabio.nci.nih.gov";
$port = "5080";
$method = "getGenes";
$searchRec{"symbol"} = "pTEN";
$s = SOAP::Lite
   -> uri($URI)
   -> proxy("http://$server:$port$PROXY_PATH");
# make service request
$som=$s->$method(SOAP::Data->type(map => \%searchRec));
# interpret result
if ($som->fault) {
   print    "FAULT    ENCOUNTERED!\nfaultcode:\t"    .    $som->faultcode    .
"\nfaultstring:\t" .
   $som->faultstring . "\n";
} else {
   $xmldoc = $som->result;
   open (OUT, ">pTEN.xml");
   print OUT $xmldoc;
   print  $xmldoc;
   close OUT;
}
```

1.5.3.5 The XML Output and the Additional Arguments

The resulting XML output of this script contains information relating to the selected gene whose symbol was specified as "PTEN." Simple features like the gene's name, title, and cross-referencing IDs into other databases are represented directly, as they are simple text strings. But by default, more complex return values that reference other caBIO objects such as *Chromosome, ExpressionFeature,* and *Taxon* are encoded as *XLinks* only.

There are two ways to retrieve further information about these embedded XLinks. The simple approach of editing your perl scripts to recursively embed the *Xlink:Href* URIs and retrieving the output may become tedious when those queries in turn return additional XLinks.

Alternatively, if you know in advance which Xlinks you will need to expand, you can do so on the first pass by adding two more arguments: *fillInObjects* and *returnHeavyXML***.** The *fillInObjects* option accepts comma-separated arguments, which specify which tags are to be opened up further. The corresponding XLinks are then "filled in" with their XML content one level deep. The *returnHeavyXML* option opens *all* the embedded XLinks one level deep.

For example, suppose you want to open up the XLinks corresponding only to *ExpressionFeature* and *MapLocation*. The syntax for this would be:

```
        $searchRec{"fillInObjects"} = "ExpressionFeature,MapLocation";
```

where *$searchRec* is the hashtable variable used to store different arguments to map to the SOAP service. Alternatively, if you want to "fill up" all the first-level XLinks in the resulting XML, use:

```
                    $searchRec{"returnHeavyXML"}="true" ;
```

You can also use these options at the command line, e.g.:

```
geneClient.pl cabio.nci.nih.gov 5080 getGenes -symbol PTEN –fillInObjects
ExpressionFeature,MapLocation
```

A number of additional Perl script examples are included in the caBIO_Demo.zip file, available at the caBIO technical resources website. Download this demonstration file, and explore the PerlSOAP subdirectory. To get a complete list of the options available for the various services and methods, refer to APIs for the classes implementing these services, on the NCICB Webservices Java Doc pages.

1.6 The caBIO HTTP Interface

1.6.1 Overview

The Hypertext Transfer Protocol (HTTP) is a generic, stateless application protocol for distributed, collaborative information systems. The HTTP uses the concept of reference provided by the Uniform Resource Identifier (URI) as a location (URL) or name (URN) for indicating the resource on which a method is to be applied. The HTTP is said to be *connectionless* because once the server has responded to the single request, the connection is dropped. Because an HTTP server treats each request as unprecedented, it is also called a *stateless* protocol.

The *n*-tier model of the caBIO architecture in Figure 1.6.1 emphasizes those components in the Presentation Layer that implement the HTTP interface. The HTTP interface utilizes web browsers such as Netscape 4+ and IE 4+. HTTP requests, which are issued as URLs on the client browsers, are processed by Java servlets on the caBIO web server, and forwarded as messages via RMI to the Object Layer.

Inside the Object Layer, the domain objects and their associated infrastructure classes (*SearchCriteria* and *SearchResult* objects) are used to register the requests and hold the data as it is fetched from the Data Layer via the data access objects. These results are then XML-encoded by the domain objects' *toXML()* methods and returned to the servlets in the Presentation Layer via RMI.

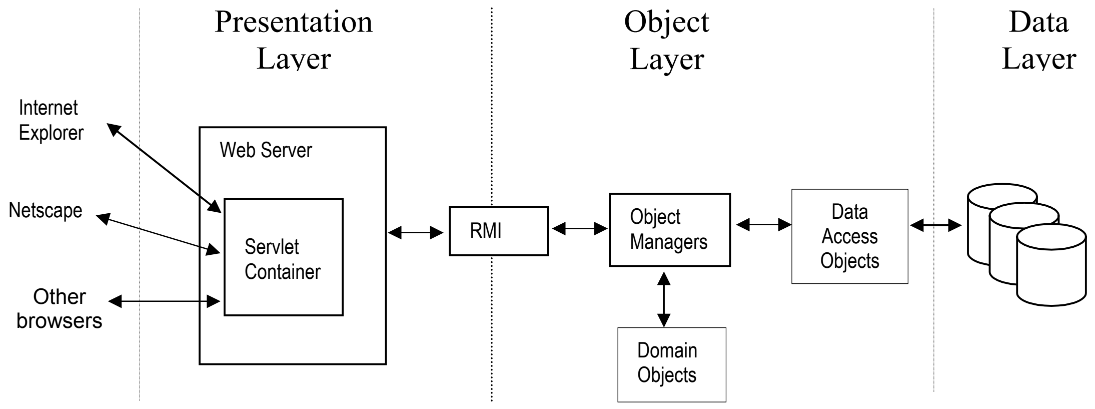


Figure 1.6.1. The caBIO HTTP interface.

Non-programmers can transform the XML-encoded HTTP response using XSL/XSLT**.** XSL (extensible style sheet language) is a language for expressing style sheets; XSL Transformations (XSLT) is a language for transforming XML documents using XSL. An example of how to use the XSL to transform an XML document is provided at the end of this section.

The caBIO objects use two devices to limit the amount of information that will be returned on a single HTTP request. The first of these is a *throttling* mechanism that limits the *number* of items returned. For example, a request to retrieve all known genes on the human genome could potentially retrieve over 30,000 gene records. To protect naïve users as well as the system from the onslaught of data that would ensue, a (re-settable) default maximum of 1,000 records per data request is enforced.

The second device limits the extent of data that will be contained in a single record, and serves as a safeguard against infinite recursion. Many of the data objects contain or otherwise entail "embedded" objects. For example, a *Gene* references the set of *Sequences* it encodes, and each of those sequences in turn references the *Gene*. Clearly, allowing each of these objects to return a full encoding of their nested references would be disastrous.

Generally, it is unlikely that the user will want access to *all* of the detailed information associated with each retrieved object, but would rather selectively specify where to drill down. The XML linking language (XLink) provides just the type of mechanism that is needed to address this concern.

The caBIO web server returns only those features that can be expressed as simple data types (i.e., strings, numbers) in the top-level encoding. Other features, such as embedded objects and arrays or other data structures, are returned as Xlinks, which specify the URL to use in a subsequent request in order to retrieve that information. As described below, however, it is also possible to selectively expand these Xlinks in the initial HTTP request using additional arguments.

1.6.2 Using the HTTP Interface

Requests sent by the client via HTTP are processed by a servlet called *getXML* residing on the caBIO server. *getXML* anticipates a list of parameters, which specify the type of object to retrieve along with additional search criteria to narrow the search. For example, the following HTTP request retrieves all genes having "PTEN" as their symbol:

```
http://cabio.nci.nih.gov:5080/CORE/GetXML?operation=Gene&Symbol=PTEN
```

The most important parameter here is *operation*, which specifies the class name of the type of domain object being requested; all other parameters are defined according to this first one. In particular, only the *Gene* objects have a *Symbol* field. Thus, if the parameters had been:

```
operation=Chromosome&Symbol=PTEN
```

an error would be produced. Referring back to Figure 1.6.1, recall that each HTTP request is effectively forwarded to the appropriate objects in the Object Layer. Accordingly, any additional criteria the user may wish to specify to narrow the search must be understandable by the receiving objects in the Object Layer.

To see the list of services that can be invoked through the HTTP interface, refer to the Java Docs page for the *webservices* package. To see the list of class names that can serve as values for the *operation* parameter, refer to the Java Docs page for the *bean* package. Example class names are: *Gene, Library, Chromosome, Sequence, etc*. To get a list of the additional parameters that can follow a given *operation* request, access the associated *SearchCriteria* object's *Set* methods in the *bean* package. Each domain object (*Xxx*) in the *bean* package has an associated search criteria object, named *XxxSearchCriteria.*

For example, the *Chromosome* domain object has an associated *ChromosomeSearchCriteria* object. While there is no explicit association between the two objects, the search criteria object is designed for use as the single argument to the domain object's *search()* method. The idea is to set a few attributes in the search criteria so as to limit the search, and to subsequently invoke the domain object's search method on that search criteria object.

Thus, we must consult the search criteria's *Set* methods to determine what attributes are settable. The *ChromosomeSearchCriteria* has only two such named methods: *SetName()* and *SetId()*. Accordingly, if we are using *Chromosome* as the value for *operation*, then the only additional legitimate parameters are *name=* and *id=*. In other words, the additional allowed parameters for the HTTP request are defined by removing the "set" prefix from the search criteria's methods.

In summary, the *operation* parameter is **mandatory;** its value must be the class name of a domain object in the *gov.nih.nci.caBIO.bean* package. All of the parameters to *GetXML* are case-sensitive — using "Operation" instead of "operation" in the HTTP request will produce an error. The class names used as values are also case-sensitive. For example, to get one or more genes, you must use *operation=Gene* – **not** *operation=gene*. Finally, in addition to specifying an *operation=* parameter, each HTTP request must include *at least one* search criterion that can be associated with the requested object type. The allowed criteria for a given object type are determined by the associated *SearchCriteria* object's *Set* methods. Refer to the Java Docs to find the domain object class names and their corresponding *SearchCriteria's* methods.

1.6.3 Drilling Down Through Xlinks

The output returned by the caBIO server in response to an HTTP request is formatted XML with embedded XLinks. There are two special request parameters that can further expand these XLinks in the XML output. The *returnHeavyXML* parameter will open up all of the embedded XLinks one level deep. For example, to search for genes whose symbol match "PTEN" and open up all Xlinks, you would use:

```
operation=Gene&symbol=PTEN&returnHeavyXML=1
```

The other parameter, *fillInObjects,* "fills" in only those XLinks whose tags are specified in a comma-separated list**.** Thus, to open up only the *GoOntology* and *ExpressionFeature* tags in the XML output, you would use:

```
operation=Gene&symbol=pTEN&fillInObjects=GoOntology,ExpressionFeature
```

1.6.4 Controlling the Number of Items Returned

It is also possible to fine-tune the default "throttling" mechanism defining the number of results returned on any single request. For example, assuming the search request yields, say, 500 results, specifying: *resultStart=450,* will return only the last 50. Similarly, one can use *resultCount=50* to get back only the first 50:

```
operation=Gene&symbol=pTEN&resultCount=50
```

Alternatively, you can use the parameter *ReturnCount* to specify the number of results to return, without concern for starting or ending indices. By default, the return results start at index 1 and the maximum number returned is 1,000.

1.6.5 Specifying the IP Address and Port in the URL

To use the HTTP API you need additional information such as the NCICB server and listening port for HTTP requests. The complete syntax of the HTTP request is:

```
http://<server:port>/CORE/GetXML?<Argument list>
```

where `<server:port>` is the caBIO web server and port number reserved for HTTP requests. These server addresses and port numbers change from time to time; check with the NCICB help desk for the current specifications.

1.6.6 Applying XSL to XML Output

Some browsers (e.g., Netscape) cannot process XML-formatted documents, and on these platforms you will need to transform the XML response to an HTML document. As mentioned in the foregoing section, XSL/XSLT can be used for these purposes.

One option is to save the XML output you receive to a file and subsequently apply an appropriate XSL style sheet. Alternatively, you can use the *ApplyXSLT* servlet running on the caBIO web server to transform the XML output in real time.

ApplyXSLT requires two parameters: *mURL* and *xslURL*. *mURL* is a "modified" URL in which the original HTTP request is modified such that all instances of "?" are replaced with "$", and all ampersands ("&") are replaced with "@". For example, the original HTTP request

```
http://cabio.nci.nih.gov/CORE/GetXML?operation=Gene&Symbol=vegf
```

would now become:

```
http:// cabio.nci.nih.gov/CORE/GetXML$operation=Gene@Symbol=vegf
```

These modifications allow the new URL to be embedded inside a second URL. This is needed because the HTTP request will now be sent directly to the *ApplyXSLT* servlet, with the original request as an argument to that servlet. *ApplyXSLT* will then issue the original HTTP request, receive the results on behalf of the client, and transform the results using the stylesheet specified in the *xslURL* parameter. The complete syntax of using *ApplyXSLT* is:

```
http://<server:port>/CORE/ApplyXSLT?mURL=<mURL>&xslURL=<xslURL>
```

For example:

http://cabio.nci.nih.gov:5080/CORE/ApplyXSLT?mURL=http://cabio.nci.nih.gov:5080/CORE/GetXML$operation=Gene@Symbol=pten&xslURL=http://cabio.nci.nih.gov/CORE//xsl/cabio-beans.xsl

1.7 caBIO Data Sources

The caBIO application programming interfaces were developed primarily in response to the need for programmatic access to the information at several NCI websites, including CGAP, CMAP, and GAI. While all of these sites provide information and search tools relevant to the molecular analysis of cancer, each organizes its information somewhat differently, emphasizing for example, gene sequences, clone libraries, chromosome maps, DNA microarray data, or clinical trials data. The primary operation in the caBIO APIs involves defining an object type of interest along with a set of search criteria for that object type, and retrieving all instances of that object that satisfy the defined search criteria.

For example, the goal might be to find all genes that are expressed in bone marrow cells, where those genes are also known to participate in apoptosis. Using the Java API, the user would first instantiate a *Gene* object and a *GeneSearchCriteria* object. The methods *setFunctionalPathway()* and *setTissueType()*, associated with *GeneSearchCriteria* objects, could then be applied to define the search criteria. A subsequent call to *myGene.search(myCriteria)* would then retrieve all genes known to satisfy these criteria.

While this information is in theory available from multiple public sites, the number of links to traverse, and the extent of collation that would have to be performed is daunting. The CGAP, CMAP, and GAI websites have distilled this information from both internal and public databases, and the caBIO data warehouses have optimized it for access with respect to the types of queries defined in the caBIO APIs. In this section we discuss the external and internal data sources for caBIO and how the information these sources provide can be accessed via caBIO objects.

The caBIO objects fall roughly into two categories: those that pertain to clinical trials data, and those that are relevant to basic research. The two groups are not mutually exclusive, as some objects such as *Gene*, *Organ,* and *Histopathology* occur in both. In particular, the *Gene* object functions as a central hub of the basic research objects and, accordingly, serves as a portal between the two relatively disjunct groups.

Before discussing the specific data sources whose information is made available via the caBIO objects, it is useful to consider the types of data that might be needed to investigate the molecular basis of cancer. The challenges are to discover which chromosome aberrations, DNA mutations, and single nucleotide polymorphisms may lead to or be associated with neoplasm formations and/or cancerous preconditions, as well as what genetic idiosyncracies may effect variable responses to treatment.

Clearly, sequence information must be available, including whole genomic sequences, expressed mRNA sequences, expressed sequence tags (ESTs), and single nucleotide polymorphisms (SNPs). Moreover, this sequence information must be available from multiple sources, including both normal and diseased tissue, so as to allow statistical analysis and identification of significant correlations. But it is not enough to provide the sequence data alone, devoid of any source information. In particular, it must be possible to identify the tissue types, histological states, and preparation methods of the samples, as well as the protocols used in generating the libraries from which the sequences were extracted. As depicted in Figure 1.7.1, *Clone* objects can be accessed directly from either a *Gene, Sequence*, or *SNP* object. The *Clone* object, in turn, provides access to information about the protocol and preparation methods for its associated *Library*, as well as access to *TraceFile* objects.

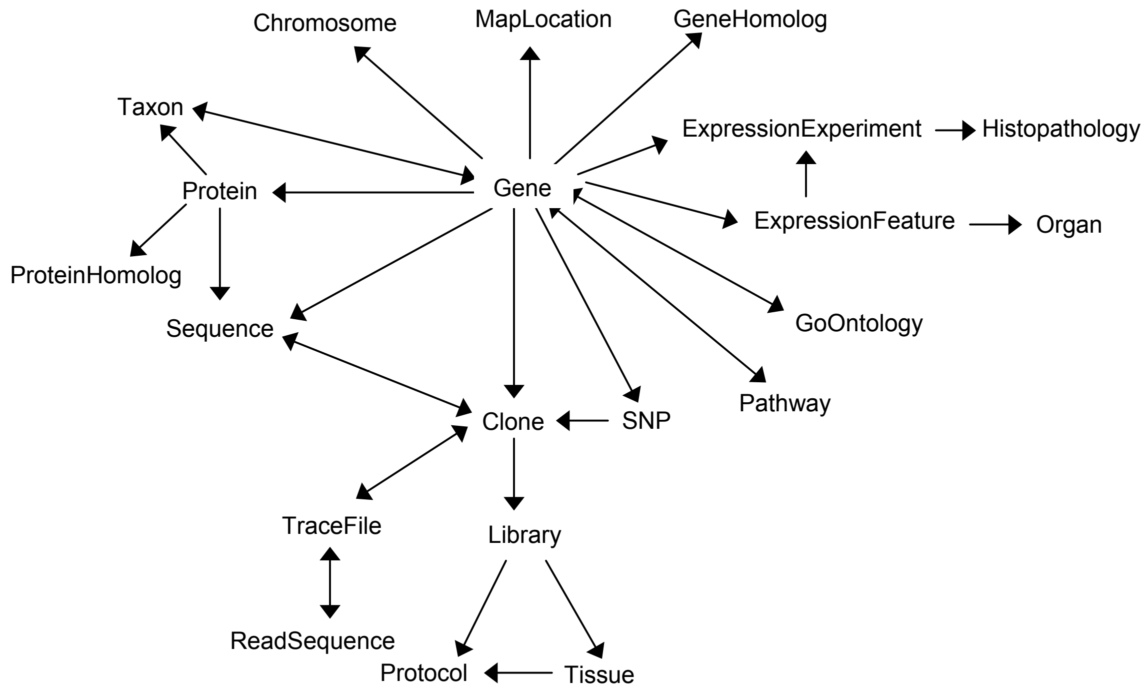


Figure 1.7.1. caBIO objects supporting basic research.

Figure 1.7.1 is a very reduced view of the entire collection of caBIO objects, showing only those objects that are most relevant to basic research. The links between objects in Figure 1.7.1 reflect only the *get* methods defined for those objects. For example, a *Library* object has *getTissue()* and *getProtocol()* methods; neither the *Tissue* nor *Protocol* objects have a *getLibrary()* method, however, so the links are unidirectional. Each object also provides access to a wealth of additional information not shown in Figure 1.7.1.

Thus, an application that attempts to identify new SNPs might use the *Clone* object to gain access to *TraceFiles*. Alternatively, an application that attempts to correlate known SNPs with disease states might use the *Clone* objects to filter SNPs according to tissue type, preparation method, and library protocol.

Chromosome and map location information are important to studies that focus on the study of oncology at the cytogenetic level. Given a chromosome that is known to have aberrations associated with cancer, this information can be used to drill down to the molecular level using the caBIO objects and appropriate search criteria on the chromosomes and map locations of genes and sequences.

Another common focus is on proteomic pathways, for example cell cycle control. The caBIO *Pathway* objects provide methods to selectively retrieve the genes occuring on that pathway, according to whether they are mutated, overexpressed, or underexpressed, etc. Thus, starting from a *Pathway* that is hypothesized to be involved in some disease etiology, it is possible to first retrieve the associated *Gene* objects and, subsequently, explore the features of each *Gene*, including its chromosome and map locations, variable expression levels (via *ExpressionExperiment* objects), and its position in the Gene Ontology Consortium hierarchies (via its *GoOntology* objects).

With the wealth of bioinformatic data that has emerged over the past decade, the need for translational research that can deliver these advances in knowledge and understanding to the clinical setting has become increasingly critical. Thus, another important type of information that must be available is clinical data.

The caBIO objects that are geared to clinical research form a clique or subgrouping among the larger set of objects and are displayed separately in Figure 1.7.2. Objects that appear in both "sub-networks" include the *Gene, Organ*, and *Histopathology* objects.

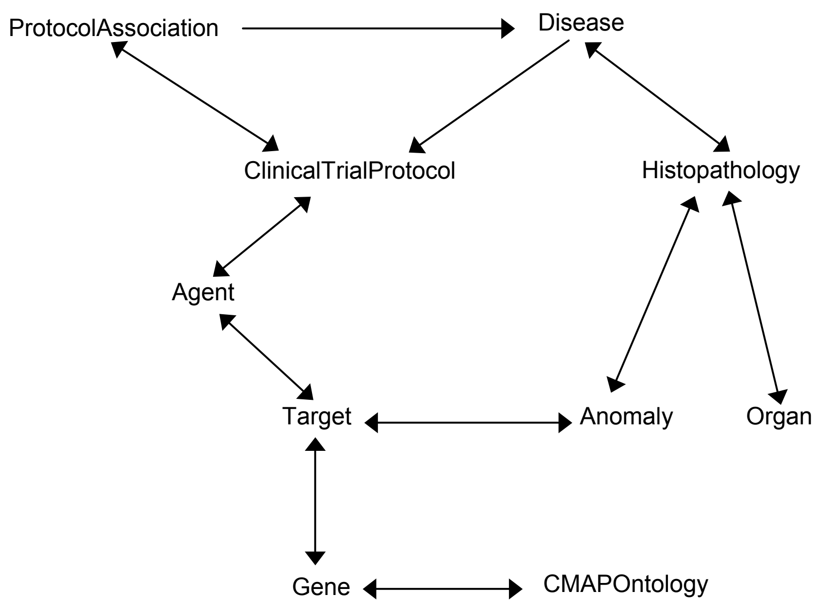


Figure 1.7.2. caBIO objects supporting clinical research.

The remainder of this section discusses the external and internal data sources whose information is used to populate the objects in Figures 1.7.1 and 1.7.2. While the caBIO data are extracted from many sources that include information from a wide variety of species, we emphasize that *only genomic data pertaining to human and mouse are available from caBIO.* caBIO provides access to curated data from multiple sources, including:

- The NCBI UniGene database [1 – 3]. Unigene provides a non-redundant partitioning of the genetic sequences contained in GenBank into gene clusters. Each such cluster has a unique UniGene ID and a list of the mRNA and EST sequences that are subsumed by that cluster. Related information stored with the cluster includes tissue types in which the gene has been expressed, mapping information, and the associated LocusLink, OMIM, and HomoloGene IDs, thus providing access to related information in those NCBI databases as well. Because the information in UniGene is centered around genes, access to Unigene is provided via the caBIO *Gene* objects. Specifically, the method *getClusterId()* associated with a *Gene* object can be used to fetch the gene's UniGene ID. Similarly, the database IDs for the NCBI OMIM and LocusLink databases can be obtained using the *getOMIMId()* and *getLocusLinkId()* methods. While there is no explicit caBIO object corresponding to a Unigene cluster, all of the information associated with the cluster is available directly via the caBIO *Gene* object's methods. For example:

- *getGenomicSequences()* returns an array containing the mRNA and EST sequences contained in the Unigene cluster;
- *getExpressionFeature()* returns an *ExpressionFeature* object, which can in turn be queried to obtain a list of the tissues in which the gene is expressed;
- *getGeneHomologs()* returns an array of *GeneHomolog* objects for the gene;
- *getChromosome()* returns the *Chromosome* on which this gene occurs;
- *getMapLocation()* returns an array of *MapLocation* objects associated with the gene.

In all of the above methods, the returned value is itself a caBIO object. Thus, further information associated with the returned object can in turn be accessed using that object's methods.

The information stored with an *ExpressionFeature* object requires a bit more explanation, as it is not actually a copy of what is stored in Unigene. caBIO's expression information is instead derived as the result of passing the Unigene free text information through a controlled vocabulary that defines only about 55 tissue types. Using an ontology to match the Unigene terms to terms in the caBIO vocabulary, the result is generally a condensed version, as several terms in the Unigene data may map to the same more general term in the vocabulary.

- NCBI's LocusLink database [4, 5]. LocusLink contains curated sequence and descriptive information associated with a gene. Each entry includes information about the gene's nomenclature, aliases, sequence accession numbers, phenotypes, UniGene cluster IDs, OMIM IDs, gene homologies, associated diseases, map locations, and a list of related terms in the Gene Ontology Consortium's ontology. Sequence accessions include a subset of GenBank accessions for a locus, as well as the NCBI Reference Sequence. As mentioned above, a caBIO Gene object has explicit methods for retrieving the gene's associated LocusLink, OMIM, and Unigene IDs. The methods to access the gene's reference sequences and aliases are *getReferenceSequences()* and *getAliases(),* respectively. Related terms in the GO ontology are retrieved using the gene's getGoOntologies() method (see discussion below). Finally, a *Gene* object's *getLocusLinkSummary()* method returns a free-text paragraph summarizing gene function.
- The Gene Ontology Consortium [6, 7]. The Gene Ontology Consortium provides a controlled vocabulary for the description of molecular functions, biological processes, and cellular components of gene products. The terms provided by the consortium define the recognized attributes of gene products and facilitate uniform queries across collaborating databases. The caBIO *Gene* object's *getGoOntologies()* method returns a list of *GoOntology* objects for the gene, which can in turn be queried to examine relationships among genes and other terms in the gene ontology.

In general, each gene is associated with one or more biological processes, and each of these processes may in turn be associated with many genes. In addition, the GO ontologies define many parent/child relationships among terms. For example, a branch of the ontology tree under `biological_process` contains the term `cell cycle control`, which in turn bifurcates into the "child" terms `cell cycle arrest`, `cell cycle checkpoint`, `control of mitosis`, etc. caBIO's *GoOntology* objects capture these relations via the *getChildRelationships(), getParentRelationships(), getOntologyHomoSapienGenes(),* and *getOntologyMouseGenes()* methods. Thus, it is possible to start with a *Gene* object and retrieve its *GoOntology* objects, and, from there, traverse a network of related genes via the links deriving from the ontological terms.

As mentioned above, caBIO does not extract ontology terms directly from the Gene Ontology Consortium but, instead, extracts those terms stored with the LocusLink entry for that gene.

- The HomoloGene database [8]. HomoloGene is an NCBI resource for curated and calculated gene homologs. The caBIO data sources capture only the calculated homologs stored by HomoloGene. These calculated homologs are the result of nucleotide sequence comparisons performed between each pair of organisms represented in UniGene clusters. The caBIO *Gene* method to access the gene's homologs is *getGeneHomologs(),* and returns an array of *GeneHomolog* objects.

- BioCarta pathways. BioCarta and its Proteomic Pathway Project (P3) provides detailed graphical renderings of pathway information concerning adhesion, apoptosis, cell activation, cell signalling, cell cycle regulation, cytokines/chemokines, developmental biology, hematopoeisis, immunology, metabolism, and neuroscience. NCI's CMAP website captures pathway information from BioCarta, and transforms the downloaded image data into Scalable Vector Graphics (SVG) representations that support interactive manipulation of the online images. The CMAP website displays BioCarta pathways selected by the user and provides options for highlighting *anomalies*, which include under- or overexpressed genes as well as mutations.

  The caBIO Pathway objects make this same information available via their associated methods, which include: *getGenes(), getExpressedGenes(), getMutatedGenes(), getOverExpressedGenes(), getUnderExpressedGenes(),* and *getTargetGenes()*. The pathway diagram is also available, as an XML document (getPathwayDiagram()) or in SVG format (*getSvgPathwayDiagram()*). The expression and mutation information that is associated with the Pathway object is derived from EST and SAGE expression data that have been culled by the CGAP project. Information about target genes is taken from data stored with CMAP.

- The Distributed Annotation System (DAS) [9] at UCSC. DAS is a client-server system that allows a single client machine to collect genome annotation information from multiple distant servers, collate the information, and display it in a single view, with little or no coordination among the information providers. DAS/1 servers are currently running at WormBase, FlyBase, Ensembl, TIGR, and UCSC. caBIO provides access to the DAS information at UCSC, via the caBIO objects defined in the *gov.nih.nci.caBIO.util.das* package.

  The starting point for any DAS search is one of the three DAS search criteria objects: *DasTypeSearchCriteria*, *DasDnaSearchCriteria,* and *DasGffFeatureSearchCriteria*. The first of these can be used to obtain an array of annotation types, the second fetches DNA sequences, and the last retrieves annotations satisfying the specified criteria. Further documentation on these search criteria objects and their affiliated domain objects can be found in the JavaDocs pages for the *das* package.

- The Cancer Genome Anatomy Project [10 – 13] (CGAP). The NCI CGAP website provides a collection of gene expression profiles of normal, pre-cancer, and cancer cells taken from various tissues. The CGAP interface allows the user to browse these profiles by various search criteria, including histology type, tissue type, library protocol, and sample preparation methods. The goal at NCI is to exploit such expression profile information for the advancement of improved detection, diagnosis, and treatment for the cancer patient. Researchers have access to all CGAP data and biological resources for human and mouse, including ESTs, gene expression patterns, SNPs, cluster assemblies, and cytogenetic information.

The CGAP website provides a powerful set of interactive data-mining tools to explore these data, and the caBIO project was initially conceived as a programmatic interface to these tools and data. Accordingly, most of the data that are available from CGAP can also be accessed through the caBIO objects. Exceptions are those data sets having proprietary restrictions, such as the Mitleman Chromosome Aberration database.

The caBIO *ExpressionExperiment* object provides a generic interface to the CGAP expression data, with methods including: *getType(), getExpressables(), getExpressionLevel(),* and *getHistopathologies()*. The type information returned by the first method is simply a string with the value “SAGE” or “EST.” *Expressable* is a Java interface that is currently implemented only by the caBIO *Gene* objects. Thus the *getExpressables()* method of an *ExpressionExperiment* returns the set of *Gene* objects whose expression levels were detected. *getExpressionLevel()* returns an array of *ExpressionLevel* objects, where each of these objects in turn provides information about the observed expression ratio for that gene. *getHistopathologies()* returns an array of *Histopathology* objects. A *Histopathology* object provides information about the organ and disease where the pathology was observed, along with information about the type of anomaly (mutation or variation in expression) associated with the histopathology.

Two subclasses, *ESTExperiment* and *SAGEExperiment,* inherit their methods from the *ExpressionExperiment* object and provide access to CGAP’s EST and SAGE data, respectively. The immediate source for EST library metadata (who made it, how many sequences were submitted, tissue, histology, other keywords, etc.) is a custom import from NCBI's dbEST database. Most of this information is also available to the public through an HTTP request at NCBI's UniGene pages, *e.g.*:

http://www.ncbi.nlm.nih.gov/UniGene/lib.cgi?ORG=Hs&LID=289

Assignment of the individual ESTs to genes is obtained from the standard UniGene dump. CGAP’s SAGE data are derived from a collaboration between NCI and Duke University and are based on new algorithms for mapping sequences to tags [13].

The caBIO *Gene* object provides access to these expression objects via its overloaded *getExpression()* method. With no arguments, this method returns an array of all SAGE and EST expression experiments for the gene. If a *type* argument (“SAGE” or “EST”) is provided, then only the experiments of that type are returned. Finally, it is also possible to specify the particular *Organ* and *Disease* of interest.

CGAP also provides access to lists of sequence-verified human and mouse cDNA IMAGE clones supplied by Invitrogen. Starting with a caBIO *Gene* object, you can get the list of *Clones* encoding that gene via the *getSequenceVerifiedClones()* method. From the *Clone* object, one can retrieve the *Library* that contains that clone using *getLibrary()*.

Specific information about the library can then be extracted using the methods *getCloneProducer(), getCloneVector(), getDescription(), getKeyword(), getLabHost(), etc.*

- The CGAP Genetic Annotation Initiative [14] (GAI). GAI is an NCI research program to explore and apply technology for identification and characterization of genetic variation in genes important in cancer. The GAI utilizes data-mining to identify "candidate" variation sites from publicly available DNA sequences, as well as laboratory methods to search for variations in cancer-related genes. All GAI candidate, validated, and confirmed genetic variants are available directly from the GAI website, and all validated SNPs have been submitted to the NCBI dbSNP database as well.

  SNPs identified by the GAI project can be accessed using caBIO *SNP* objects. The *SNP* object provides access to the *Clones* in which the SNP was observed via the *getClones()* method. The offset of this SNP in the parent sequence is available from the *getOffset()* method. The two most common base substitutions occurring at the site are extracted using *getBase1()* and *getBase2()*. The *getScore()* method returns the confidence score for the predicted SNP, and *getTracefiles()* provides access to the trace files used to identify the site as an SNP. The sequencing trace files used by GAI are imported from Washington University.

- The NCI Cancer Therapy Evaluation Program [15] (CTEP). CTEP funds an extensive national program of basic and clinical research to evaluate new anti-cancer agents, with a particular emphasis on translational research to elucidate molecular targets and drug mechanisms. In response to this emergent need for translational research, there has been a groundswell of translational support tools defining controlled vocabularies and registered terminologies so as to enhance electronic data exchange in areas that have heretofore been relatively non-computational. The caBIO trials data is updated with new CTEP data on a quarterly basis, and many of the objects in Figure 1.7.2 are designed to support translational research.

  For example, a caBIO *Target* object represents a molecule of special diagnostic or therapeutic interest for cancer research, and an *Anomaly* object is an observed deviation in the structure or expression of a *Target*. An *Agent* is a drug or other intervention that is effective in the presence of one or more specific *Targets*. The *ClinicalTrialProtocol* object organizes administrative information pertaining to that protocol and has a *getAgents()* method for programmatic access to the specifc therapies deployed.

- NCI's Cancer Molecular Analysis Project (CMAP) [16] website is powered by caBIO, and makes extensive use of the objects in both Figures 1.7.1 and 1.7.2. The goal of CMAP is to enable researchers to identify and evaluate molecular targets in cancer. Towards this goal, CMAP provides four interfaces.

  The CMAP *Profile Query* tool finds genes with the highest or lowest expression levels (using SAGE and microarray data) for a given tissue and histology. Selecting a gene from the resulting table then leads to a *Gene Info* page, providing information about cytogenetic location, chromosome aberrations, protein similarities, curated and computed orthologs, and sequence-verified as well as full-length MGC clones, along with links to various other databases. The CMAP ontology can be accessed through the caBIO *CMAPOntology* object.

CMAP's *Molecular Targets* interface organizes collections of genes by pathways and by ontology. Two ontologies are available: (1) the GO ontology described above, and (2) the CMAP ontology described here. The CMAP ontology relates functional classifications to molecular targets and agents. For example, selecting "angiogenesis" as the functional term brings up `KDR`, a type III receptor tyrosine kinase, and a list of agents for `KDR`. Selecting the target then produces a *Gene Info* page providing information about cytogenetic location, chromosome aberrations, protein similarities, curated and computed orthologs, and sequence-verified as well as full-length MGC clones, along with links to various other databases. The CMAP ontology can be accessed through the caBIO *CMAPOntology* object.

CMAP's *AgentSearch* tool allows the researcher to search for drug therapies by name (with wildcard matching), with the option of restricting the search to agents that are either associated with a term in the CMAPOntology or registered with a CTEP protocol. If the agent is associated with CTEP protocols, a table is presented on the *Agent Info* page, listing the title of each protocol and a link to its associated documentation. Selecting an entry from this table in turn leads to the *Therapeutic Trials Info* page for that CTEP protocol.

With the exception of the Mitelman Chromosome Aberration data, all of the information available through CGAP is also accessible programmatically through the caBIO objects in Figures 1.7.1 and 1.7.2.

- The NCI Enterprise Vocabulary Services [17] (EVS) The EVS is a part of the caCORE project, and is described in the next chapter. The EVS provides NCI with services and resources for controlled biomedical vocabularies. The EVS includes the NCI Thesaurus and the NCI Metathesaurus. The Thesaurus is composed of over 20,000 concepts represented by about 80,000 terms. The Thesaurus is organized into 17 hierarchical trees covering areas such as Neoplasms, Drugs, Anatomy, Genes, Proteins, and Techniques. These terms are deployed by NCI in its automated systems for uses such as keywording and database coding.

  The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing, and data pooling for clinical trials and scientific databases. The Metathesaurus is based on the NLM's Unified Medical Language System (UMLS) and is composed of over 70 biomedical vocabularies.

  Integration of the EVS Vocabulary models and the caBIO domain objects is in progress, with plans to fully specify and implement formally defined relationships between the vocabulary model's semantic concepts and the caBIO objects. Figure 1.7.3 illustrates the three Java objects defined in a programmtic interface to the EVS. These classes are implemented in a separate java package (*gov.nih.nci.caBIO.evs*) with an interface to the caBIO domain objects (implemented in *gov.nih.nci.caBIO.bean*) via the Agent class.

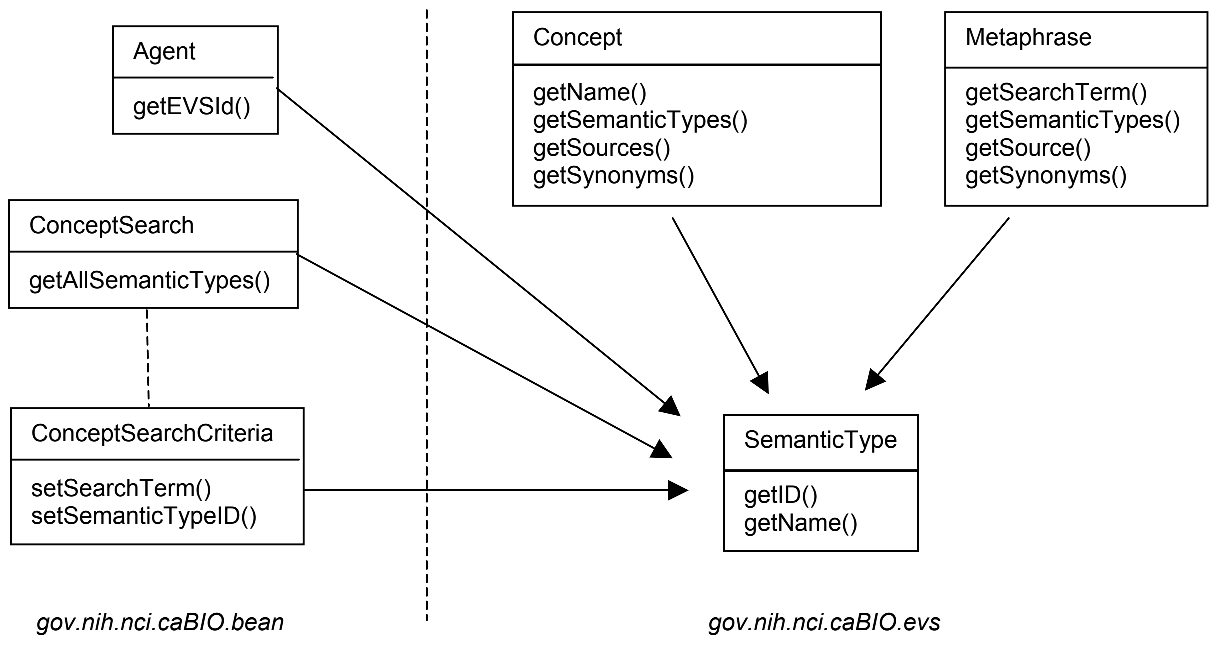


Figure 1.7.3. The caBIO Java EVS package.

1.8 References

1. Schuler (1997). Pieces of the puzzle: expressed sequence tags and the catalog of human genes. J Mol Med 75(10):694–698.
2. Schuler et al. (1996). A gene map of the human genome. Science 274: 540–546.
3. Boguski & Schuler (1995). ESTablishing a human transcript map. Nature Genetics 10: 369–371.
4. Pruitt KD, and Maglott DR (2001). RefSeq and LocusLink: NCBI gene-centered resources. Nucleic Acids Res 29(1):137–140.
5. Pruitt KD, Katz KS, Sicotte H, and Maglott DR (2000). Introducing RefSeq and LocusLink: curated human genome resources at the NCBI. Trends Genet.16(1):44–47.
6. The Gene Ontology Consortium. (2000). Gene ontology: tool for the unification of biology. Nature Genetics 25:25–29.
7. The Gene Ontology Consortium. (2001). Creating the gene ontology resource: design and implementation. Genome Research 11:1425–1433.
8. Zhang, Schwartz, Wagner, and Miller (2000). A Greedy algorithm for aligning DNA sequences, J. Comp. Biol. 7(1-2):203–14.
9. Dowell RD, Jokerst RM, Day A, Eddy SR, Stein L. The Distributed Annotation System. BMC Bioinformatics. 2001;2(1):7
10. Strausberg RL (2001). The Cancer Genome Anatomy Project: new resources for reading the molecular signatures of cancer. J Pathol, 195:31–40.
11. Strausberg RL, Buetow KH, Emmert-Buck M, and Klausner R. (2000). The Cancer Genome Anatomy Project: building an annotated gene index. Trends in Genetics, 16:103–106.
12. Strausberg RL (1999). The Cancer Genome Anatomy Project: building a new information and technology platform for cancer research. In: *Molecular Pathology of Early Cancer* (Srivastava, S., Henson, D.E., Gazdar, A., eds. IOS Press, 365–370.
13. Boon K, Osorio EC, Greenhut SF, Schaefer CF, Shoemaker J, Polyak K, Morin PJ, Buetow KH, Strausberg RL, De Souza SJ, Riggins GJ (2002). An anatomy of normal and malignant gene expression. Proc Natl Acad Sci U S A 2002 Jul 15, *in press*.
14. Clifford R, Edmonson M, Hu Y, Nguyen C, Scherpbier T, and Buetow KH (2000). Expression-based genetic/physical maps of single-nucleotide polymorphisms identified by the cancer genome anatomy project. Genome Res 10(8):1259–65.
15. Ansher SS, Scharf R (2001). The Cancer Therapy Evaluation Program (CTEP) at the National Cancer Institute: industry collaborations in new agent development. Ann N Y Acad Sci. 949:333-40.
16. Buetow KH, Klausner RD, Fine H, Kaplan R, Singer DS, Strausberg RL (2002). Cancer Molecular Analysis Project: Weaving a rich cancer research tapestry.Cancer Cell 1(4):315-8.
17. Hartel, F.W. and de Coronado, S (2002). Information Standards within NCI. In: *Cancer Informatics: Essential Technologies for Clinical Trials.* J. S. Silva, M. J. Ball, C. G. Chute, J. V. Douglas, C. Langlotz, J. Niland and W Scherlis, eds. Springer-Verlag.

1.9 The Domain Object Catalog

1.9.1 gov.nih.nci.caBIO.bean

1.9.1.1.1 Agent
A therapeutic agent (drug, intervention therapy) used in a clinical trial protocol.
Application: used primarily by CMAP and EVS applications.
Related domain objects: ClinicalTrialProtocol, Target
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.2 Anomaly
An irregularity in either the expression of a gene or its structure (i.e., a mutation).
Application: defined and used by the CMAP project.
Related domain objects: Histopathology, Target
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.3 Chromosome
An object representing a specific chromosome for a specific taxon; provides access to all known genes contained in the chromosome and to the taxon.
Application: used by CMAP and other applications to reason about the molecular basis of cancer.
Related domain objects: Gene, Taxon
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.4 ClinicalTrialProtocol
The protocol associated with a clinical trial; organizes administrative information about the trial such as Organization ID, participants, phase, etc., and provides access to the administered Agents.
Application: used primarily by CMAP.
Related domain objects: Agent, ProtocolAssociation
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.5 Clone
An object used to hold information pertaining to I.M.A.G.E. clones; provides access to sequence information, associated trace files, and the clone's library.
Application: The caBIO Clone data are imported from the CGAP website databases.
Related domain objects: Sequence, Library, TraceFile, SNP
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.6 CMAPOntology
An object providing entry to the CMAP gene ontology, which categorizes genes by function; provides access to gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree.
Application: defined and used by CMAP applications.

Related domain objects: CMAPOntologyRelationship, Gene
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Ontologable

1.9.1.1.7 CMAPOntologyRelationship
An object specifying a child or parent relationship between CMAPOntology objects.
Application: used and defined by CMAP applications.
Related domain objects: CMAPOntology
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Relationable

1.9.1.1.8 ConceptSearch
Represents a searchable concept term in a controlled vocabulary occurring in the NCI Metathesaurus; used to find synonym or semantic types for the concept of interest. Related domain objects are defined in the *evs* package.
Application: used primarily by EVS applications.
Related domain objects: `gov.nih.nci.caBIO.evs.SemanticType`
Extends: java.lang.Object

1.9.1.1.9 ConsensusSequence
A specialization of the Sequence class; represents the consensus of a set of contigs, which it also provides access to.
Application: used by the GAI project to identify SNPs.
Related domain objects: Contig, Gene, Protein, Clone, ExpressionMeasurement
Extends: Sequence
Implements: XMLInterface, java.io.Serializable

1.9.1.1.10 Contig
One of the set of overlapping sequence fragments used to assemble a consensus sequence, which it also provides access to.
Application: Used by the GAI project to identify SNPs.
Related domain objects: Sequence, ConsensusSequence
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.11 Disease
Disease objects specify a disease name and ID; disease objects also provide access to: ontological relations to other diseases; clinical trial protocols treating the disease; and specific histologies associated with instances of the disease.
Application: used by the CMAP project.
Related domain objects: ClinicalTrialProtocol, Histopathology, DiseaseRelationship
Extends java.lang.Object
Implements: XMLInterface, java.io.Serializable, Ontologable

1.9.1.1.12 DiseaseRelationship
An object specifying a child or parent relationship between Disease objects.
Application: used by the CMAP project.

Related domain objects: Disease
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Relationable

1.9.1.1.13 ESTExperiment
An object representing data from an expression experiment using expressed sequence tags.
Application: caBIO's EST experiment data are downloaded from the CGAP databases.
Related domain objects: ExpressionExperiment, Gene, Histopathology
Extends: ExpressionExperiment
Implements: XMLInterface, java.io.Serializable

1.9.1.1.14 ExpressionExperiment
A virtual class defining the methods and attributes shared by various types of expression experiments, including ESTExperiment and SAGEExperiment objects.
Related domain objects: Gene, Histopathology, ESTExperiment, SAGEExperiment
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.15 ExpressionFeature
Associated with a Gene object through the gene's method getExpressionFeature(); provides access to the list of organs where the gene is known to be expressed.
Application: Expression information for a gene is extracted from the CGAP databases, which are based on the information in Unigene (see Section 1.7).
Related domain objects: Organ, Gene.
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.16 ExpressionMeasurement
An object representing a structure that is capable of measuring the absolute or relative amount of a given compound.
Related domain objects: Gene, Sequence
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.17 ExpressionMeasurementArray
An array of ExpressionMeasurement objects.
Related domain objects: Gene, Sequence
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.18 Gene
Gene objects are the effective portal to most of the genomic information provided by the caBIO data services; organs, diseases, chromosomes, pathways, sequence data, and expression experiments are among the many objects accessible via a gene.
Related domain objects: ExpressionFeature, Organ, Disease, Chromosome, Taxon, Sequence, GeneAlias, GeneHomolog, MapLocation, Protein, SNP, Target, ExpressionMeasurement, Pathway, GoOntology

Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Expressable

1.9.1.1.19 GeneAlias
An alternative name for a gene; provides descriptive information about the gene (as it is known by this alias), as well as access to the Gene object it refers to.
Related Domain Objects: Gene
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.20 GeneHomolog
Defined only in relation to another Gene object of interest, the GeneHomolog in caBIO is the functional equivalent of that gene in another taxon (i.e., its ortholog). The GeneHomolog is a specialization of the parent Gene object; in addition to all of the methods provided by the gene interface, the homolog object provides the percent of sequence similarity of the homolog to the related gene of interest.
Related domain objects: Gene, ExpressionFeature, Organ, Disease, Chromosome, Taxon, Sequence, GeneAlias, GeneHomolog, MapLocation, Protein, SNP, Target, ExpressionMeasurement, Pathway, GoOntology
Extends: Gene
Implements: XMLInterface, java.io.Serializable, Expressable

1.9.1.1.21 GoOntology
An object providing entry to a Gene object's position in the Gene Ontology Consortium's controlled vocabularies, as recorded by LocusLink; provides access to gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree.
Related domain objects: Gene, GoOntologyRelationship
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Ontologable

1.9.1.1.22 GoOntologyRelationship
An object specifying a child or parent relationship between GoOntology objects.
Related domain objects: GoOntology
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Relationable

1.9.1.1.23 Histopathology
An object representing anatomical changes in a diseased tissue sample associated with an expression experiment; captures the relationship between organ and disease.
Application: used by the CMAP project.
Related domain objects: Anomaly, Organ, Disease, ExpressionExperiment, ESTExperiment, SAGEExperiment
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.24 Library
An object representing a CGAP library; provides access to information about: the tissue sample and its method of preparation, the library protocol that was used, the clones contained in the library, and the sequence information derived from the library.
Application: The caBIO libaries are extracted from the CGAP databases.
Related domain objects: Clone, Sequence, Tissue, Protocol
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.25 MapLocation
Associated with a Gene object, the physical map location of the gene.
Related domain objects: Chromosome, Gene, Taxon.
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.26 Organ
A representation of an organ whose name occurs in a controlled vocabulary; provides access to any Histopathology objects for the organ, and, because it is "ontolog-able," provides access to its ancestral and descendant terms in the vocabulary.
Related domain objects: Histopathology, OrganRelationship
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Ontologable

1.9.1.1.27 OrganRelationship
An object specifying a child or parent relationship between Organ objects.
Related domain objects: Organ
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable, Relationable

1.9.1.1.28 Pathway
An object representation of a molecular/cellular pathway compiled by BioCarta. Pathways are associated with specific Taxon objects, and contain multiple Gene objects, which may be Targets for treatment.
Related domain objects: Gene, Taxon, Target.
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.29 Protein
An object representation of a protein; provides access to the encoding gene via its GenBank ID, the taxon in which this instance of the protein occurs, and references to homologous proteins in other species.
Related domain objects: Gene, ProteinHomolog, Taxon.
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.30 ProteinHomolog
Defined only in relation to another Protein object of interest, the ProteinHomolog in caBIO is the functional equivalent of that protein in another taxon (i.e., its ortholog). The ProteinHomolog is a specialization of the parent Protein object; in addition to the methods provided by the protein interface, the Homolog object provides the percent of sequence similarity of the homolog to the related protein of interest.
Related domain objects: Gene, Protein, Taxon.
Extends: Protein
Implements: XMLInterface, java.io.Serializable

1.9.1.1.31 Protocol
An object representation of the protocol used in assembling a clone library.
Application:
Related domain objects: Library
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.32 ProtocolAssociation
An association class relating between ClinicalTrialProtocols to Diseases.
Application: used primarily by the CMAP project.
Related domain objects: ConceptSearch, ClinicalTrialProtocol, Disease
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.33 ReadSequence
The output of a TraceFile object, an ASCII representation of the nucleotide sequence; a read sequence is created by running PHRED.
Application: the GAI project.
Related domain objects: TraceFile, SNP
Extends: Sequence
Implements: XMLInterface, java.io.Serializable

1.9.1.1.34 SAGEExperiment
A specialization of the ExpressionExperiment class, used to represent serial analysis of gene expression (SAGE) data.
Application: The caBIO SAGE data are derived from methods developed at NCI by the CGAP project in collaboration with Duke University.
Related domain objects: ExpressionExperiment, Gene, Histopathology
Extends: ExpressionExperiment
Implements: XMLInterface, java.io.Serializable

1.9.1.1.35 Sequence
An object representation of a gene sequence; provides access to the clones from which it was derived, the ASCII representation of the residues it contains, and the sequence ID.
Related domain objects: Clone, Gene, Protein
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.36 SNP
An object representing a Single Nucleotide Polymorphism; provides access to the clones and the trace files from which it was identified, the two most common substitutions at that position, the offset of the SNP in the parent sequence, and a confidence score.
Application: The SNPs provided by caBIO were identified by the GAI project.
Related domain objects: Clone, TraceFile
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.37 Target
A gene thought to be at the root of a disease etiology, and which is targeted for therapeutic intervention in a clinical trial.
Application: defined and used by the CMAP project.
Related domain objects: Agent, Anomaly, Gene
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.38 Taxon
An object representing the various names (scientific, common, abbreviated, etc.) for a species associated with a specific instance of a Gene, Chromosome, Pathway, Protein, or Tissue.
Related domain objects: Gene, Chromosome, Pathway, Protein, Tissue
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.39 Tissue
A group of similar cells united to perform a specific function.
Related domain objects: Disease, Organ, Protocol, Taxon
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.1.1.40 TraceFile
An object representing the recorded trace file used to identify an SNP, based on the observed intensities for the four possible bases at each position in the sequence.
Application: All TraceFiles available through caBIO are from Washington University.
Related domain objects: Clone, ReadSequence, SNP
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.2 gov.nih.nci.caBIO.evs

1.9.2.1.1 Concept
An object representing a *concept* in the NCI Source vocabulary accessable through the Metaphrase browser.
Application: used primarily by CMAP and EVS applications.
Related domain objects: SemanticType
Extends: java.lang.Object
Implements: java.io.Serializable

1.9.2.1.2 Metaphrase
An object representing the NCI Metaphrase browser.
Application: used primarily by CMAP and EVS applications.
Related domain objects: SemanticType, Concept
Extends: java.lang.Object
Implements: java.io.Serializable

1.9.2.1.3 SemanticType
An object representing the semantic type of a *concept* in the NCI Source vocabulary.
Application: used primarily by CMAP and EVS applications.
Extends: java.lang.Object
Implements: XMLInterface, java.io.Serializable

1.9.3 gov.nih.nci.caBIO.util.das

All of the following domain objects were auto-generated from the DAS DTD. For definitions of the classes, refer to the specifications of the Distributed Annotation Server web site. The definitions below concern extensions to the auto-generated classes.

1.9.3.1.1 DasDnaDna
The DNA class auto-generated from the DAS DTD; modified to be serializable and to include support for calling the sequencemanager proxy.
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable
1.9.3.1.2
1.9.3.1.3 DasDnaSequence
Related domain objects: DasDnaDna
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.4 DasDnaSet
Extends: javax.xml.bind.MarshallableRootElement
Implements: javax.xml.bind.Element, javax.xml.bind.RootElement

1.9.3.1.5 DasDsn
Related domain objects: DasDsnSource
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.6 DasDsnDescription
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.7 DasDsnSet
Extends: javax.xml.bind.MarshallableRootElement
Implements: javax.xml.bind.Element, javax.xml.bind.RootElement

1.9.3.1.8 DasDsnSource
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.9 DasGff
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.10 DasGffFeature
Related domain objects: DasGffMethod, DasGffType
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.11 DasGffGroup
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.12 DasGffLink
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.13 DasGffMethod
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.14 DasGffSegment
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.15 DasGffSet
Related domain objects: DasGff
Extends: javax.xml.bind.MarshallableRootElement
Implements: javax.xml.bind.Element, javax.xml.bind.RootElement

1.9.3.1.16 DasGffTarget
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.17 DasGffType
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element, java.io.Serializable

1.9.3.1.18 DasSegment
This class is used to specify a segment of DNA during queries to a DAS server. Segments can be specified for Feature and Type searches. Segments are typically defined by a chromosone identifier, starting position and ending position.

Extends: java.lang.Object
Implements: java.io.Serializable

1.9.3.1.19 DasTypesGff
Related domain objects: DasTypesSegment
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.20 DasTypesSegment
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.9.3.1.21 DasTypesSet
Related domain objects: DasTypesGff
Extends: javax.xml.bind.MarshallableRootElement
Implements: javax.xml.bind.Element, javax.xml.bind.RootElement

1.9.3.1.22 DasTypesType
Extends: javax.xml.bind.MarshallableObject
Implements: javax.xml.bind.Element

1.10 SearchCriteria Object Mappings

As described in Section 1.3, it is possible to set the desired attributes of one SearchCriteria object and, subsequently, embed that object in a second SearchCriteria object using the *putSearchCriteria()* method.

However, not all SearchCriteria are "compatible" with one another. Column 1 in Table A.2.1. below lists those SearchCriteria that support the *putSearchCriteria()* method, and column 2 lists the secondary SearchCriteria objects that can be provided as arguments to the first object's *putSearchCriteria()* method. This list is dynamic and new entries are added as necessary.

SearchCriteria #1	*(accepts)* SearchCriteria #2
AgentSearchCriteria	ClinicalTrialSearchCriteria
AnomalySearchCriteria	DiseaseSearchCriteria, HistopathologySearchCriteria, TargetSearchCriteria, VocabularySearchCriteria
ChromosomeSearchCriteria	GeneSearchCriteria
ClinicalTrialProtocolSearchCriteria	AgentSearchCriteria, DiseaseSearchCriteria, HistopathologySearchCriteria, OrganSearchCriteria
CloneSearchCriteria	SequenceSearchCriteria, GeneSearchCriteria
ExpressionFeatureSearchCriteria	GeneSearchCriteria
ExpressionMeasurementSearchCriteria	ExpressionMeasurementArraySearchCriteria, GeneSearchCriteria, SequenceSearchCriteria
GeneAliasSearchCriteria	GeneSearchCriteria
GeneHomologSearchCriteria	GeneSearchCriteria
GeneSearchCriteria	ChromosomeSearchCriteria, CloneSearchCriteria, ExpressionMeasurementSearchCriteria, HistopathologySearchCriteria, OntologySearchCriteria, OrganSearchCriteria, PathwaySearchCriteria, SequenceSearchCriteria, TargetSearchCriteria, TaxonSearchCriteria
GoOntologySearchCriteria	GeneSearchCriteria
HistopathologySearchCriteria	DiseaseSearchCriteria, OrganSearchCriteria
LibrarySearchCriteria	GeneSearchCriteria, HistopathologySearchCriteria, ProtocolSearchCriteria
MapLocationSearchCriteria	GeneSearchCriteria
PathwaySearchCriteria	GeneSearchCriteria, HistopathologySearchCriteria
ProteinSearchCriteria	GeneSearchCriteria
SequenceSearchCriteria	CloneSearchCriteria, GeneSearchCriteria, OntologySearchCriteria
TargetSearchCriteria	AnomalySearchCriteria, CMAPOntologySearchCriteria
TaxonSearchCriteria	GeneSearchCriteria

Table A.2. 1. Object-specific *putSearchCriteria* arguments.

```
For example, the following code snippet will retrieve all pathways
containing genes whose symbols match the string “vegf”:
// define the criteria for the genes we are interested in:
GeneSearchCriteria GeneCriteria = new GeneSearchCriteria();
GeneCriteria.setSymbol("vegf");

// now define the criteria for pathways, and embed the gene criteria:
PathwaySearchCriteria PathCriteria = new PathwaySearchCriteria();
PathCriteria.PutSearchCriteria(GeneCriteria);

// create a pathway object and invoke its search method:
Pathway myPath = new Pathway();
SearchResult result = MyPath.Search(PathCriteria);
if (result != null){
    Pathway[] myPaths = (Pathway[]) result.getResultSet();
     // ... do something interesting with the paths
}
```

Criteria Attribute Maps

For convenience, we summarize here the object-specific settable attributes of the various search criteria, which can be used to narrow the search for an associated class of objects. Each of these attributes is a private data member of the class, but is settable via the *set* method of the same name.

In addition to these object-specific attributes, each search criteria object also implements the *setOrderBy()* method, which controls the order in which the results are returned.

AgentSearchCriteria	
agentNSCNumber_Attribute	java.lang.String
comment_Attribute	java.lang.String
eVSId_Attribute	java.lang.String
isCMAPAgent_Attribute	java.lang.String
name_Attribute	java.lang.String
clinicalTrialProtocolId_Attribute	java.lang.String
source_Attribute	java.lang.String
targetId_Attribute	java.lang.String
AnomalySearchCriteria	
anomalyDescription_Attribute	java.lang.String
contextCode_Attribute	java.lang.String
HistopathologyId_Attribute	java.lang.String
organId_Attribute	java.lang.String
targetId_Attribute	java.lang.String
ChromosomeSearchCriteria	
name_Attribute	java.lang.String
ClinicalTrialProtocolSearchCriteria	
agent_Attribute	java.lang.String
agentId_Attribute	java.lang.String
conceptId_Attribute	java.lang.String
ctepName_Attribute	java.lang.String

diseaseCategory_Attribute	java.lang.String
diseaseId_Attribute	java.lang.String
diseaseName_Attribute	java.lang.String
imtCode_Attribute	java.lang.String
leadOrganizationName_Attribute	java.lang.String
leadOrganizationId_Attribute	java.lang.String
nihAdminCode	java.lang.String
pdqIdentifier_Attribute	java.lang.String
phaseIdentifier_Attribute	java.lang.String
documentNumber_Attribute	java.lang.String
protocolAssociationId_Attribute	java.lang.String
piName_Attribute	java.lang.String
title_Attribute	java.lang.String
treatmentFlag_Attribute	java.lang.String

ClinicalTrialSearchCriteria

ctep_Name_Attribute	java.lang.String
disease_Category_Attribute	java.lang.String
disease_Name_Attribute	java.lang.String
imt_Code_attribute	java.lang.String
lead_Organization_attribute	java.lang.String
phase_Attribute	java.lang.String
protocol_Document_Number_Attribute	java.lang.String
protocol_Id_Attribute	java.lang.String
title_Attribute	java.lang.String

CloneSearchCriteria

geneId_Attribute	java.lang.String
sequenceId_Attribute	java.lang.String
name_Attribute	java.lang.String
snpId_Attribute	java.lang.String
verified_Attribute	java.lang.Boolean

CMAPOntologySearchCriteria

cMAPChildId_Attribute	java.lang.String
cMAPGeneId_Attribute	java.lang.String
cMAPName_Attribute	java.lang.String
cMAPParentId_Attribute	java.lang.String
cMAPOntology_Attribute	java.lang.String
includeBoth_Attribute	java.lang.String
includeParents_Attribute	java.lang.String
includeChildren_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipChildId_Attribute	java.lang.String
relationshipType_Attribute	java.lang.String

CMAPOntologyRelationshipSearchCriteria

relationshipParentId_Attribute	java.lang.String
relationshipChildId_Attribute	java.lang.String

relationshipType_Attribute	java.lang.String
ConsensusSequenceSearchCriteria	
consensusSequenceType_Attribute	java.lang.String
ContigId_Attribute	java.lang.String
geneId_Attribute	java.lang.String
proteinId_Attribute	java.lang.String
refGeneId_Attribute	java.lang.String
ContigSearchCriteria	
sequenceId_Attribute	java.lang.String
Name_attribute	java.lang.String
DiseaseSearchCriteria	
diseaseId_Attribute	java.lang.String
histopathologyId_Attribute	java.lang.String
geneId_Attribute	java.lang.String
includeBoth_Attribute	java.lang.String
includeChildren_Attribute	java.lang.String
includeParents_Attribute	java.lang.String
name_Attribute	java.lang.String
relationshipChildId_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipType_attribute	java.lang.String
DiseaseRelationshipSearchCriteria	
relationshipChildId_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipType_attribute	java.lang.String
ExpressionExperimentSearchCriteria	
gene_Attribute	java.lang.String
geneId_Attribute	java.lang.String
expressionFeatureId_Attribute	java.lang.String
organ_attribute	java.lang.String
proteinId_Attribute	java.lang.String
taxonId_attribute	java.lang.String
threshold_Attribute	java.lang.String
type_Attribute	java.lang.String
ExpressionFeatureSearchCriteria	
geneId_attribute	java.lang.String
ExpressionMeasurementSearchCriteria	
accessionNumber_Attribute	java.lang.String
expressionMeasurementArrayId_Attribute	java.lang.String
geneId_Attribute	java.lang.String
name_Attribute	java.lang.String
sequenceId_Attribute	java.lang.String

ExpressionMeasurementArraySearchCriteria

expressionMeasurementId_Attribute	java.lang.String
accessionNumber_Attribute	java.lang.String
name_Attribute	java.lang.String

GeneSearchCriteria

bcId_Attribute	java.lang.String
cloneName_Attribute	java.lang.String
organism_Attribute	java.lang.String
symbol_Attribute	java.lang.String
chromosomeId_Attribute	java.lang.String
cMAPOntologyId_Attribute	java.lang.String
cytogenicLocation_Attribute	java.lang.String
expressionMeasurementId_Attribute	java.lang.String
allPathwayId_Attribute	java.lang.String
locusLinkSummary_Attribute	java.lang.String
expressedPathwayId_Attribute	java.lang.String
functionalPathway_Attribute	java.lang.String
overExpressedPathwayId_Attribute	java.lang.String
underExpressedPathwayId_Attribute	java.lang.String
PathwayId_Attribute	java.lang.String
mutatedGenePathwayID_attribute	java.lang.String
geneBankAccessionNUmber_Attribute	java.lang.String
geneKeyword_Attribute	java.lang.String
geneNameKeyword_Attribute	java.lang.String
goOntologyMouseId_Attribute	java.lang.String
goOntologyHomoSapienId_Attribute	java.lang.String
goOntologyId_Attribute	java.lang.String
uniqueIdentifier_Attribute	java.lang.String
symbol_Attribute	java.lang.String
targetId_Attribute	java.lang.String
taxonId_Attribute	java.lang.String
tissueType_Attribute	java.lang.String
unigeneClusterId_Attribute	java.lang.String

GeneAliasSearchCriteria

description_Attribute	java.lang.String
geneId_Attribute	java.lang.String
type_Attribute	java.lang.String

GeneHomologSearchCriteria

geneId_Attribute	java.lang.String

GoOntologySearchCriteria

diseaseId_Attribute	java.lang.String
histopathologyId_Attribute	java.lang.String
geneId_Attribute	java.lang.String
includeBoth_Attribute	java.lang.String
includeChildren_Attribute	java.lang.String

includeParents_Attribute	java.lang.String
name_Attribute	java.lang.String
relationshipChildId_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipType_attribute	java.lang.String

GoOntologyRelationshipSearchCriteria

relationshipChildId_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipType_attribute	java.lang.String

HistopathologySearchCriteria

diseaseId_Attribute	java.lang.String
expressionExperimentid_Attribute	java.lang.String
name_attribute	java.lang.String
organId_Attribute	java.lang.String

LibrarySearchCriteria

geneId_Attribute	java.lang.String
libraryGroup_Attribute	java.lang.String
libraryName_Attribute	java.lang.String
libraryProtocol_Attribute	java.lang.String
tissueType_Attribute	java.lang.String
tissuePreparation_Attribute	java.lang.String
tissueName_Attribute	java.lang.String
tissueHistology_Attribute	java.lang.String
sortOrder_Attribute	java.lang.String
organism_Attribute	java.lang.String

MapLocationSearchCriteria

geneId_Attribute	java.lang.String
location_Attribute	java.lang.String
type_Attribute	java.lang.String

OrganSearchCriteria

anomaly_id	java.lang.String
expressionFeatureId_Attribute	java.lang.String
diseaseId_Attribute	java.lang.String
histopathologyId_Attribute	java.lang.String
geneId_Attribute	java.lang.String
includeBoth_Attribute	java.lang.String
includeChildren_Attribute	java.lang.String
includeParents_Attribute	java.lang.String
name_Attribute	java.lang.String
relationshipChildId_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipType_attribute	java.lang.String

OrganRelationshipSearchCriteria	
relationshipChildId_Attribute	java.lang.String
relationshipParentId_Attribute	java.lang.String
relationshipType_Attribute	java.lang.String
PathwaySearchCriteria	
context_Attribute	java.lang.String
displayValue_Attribute	java.lang.String
geneId_Attribute	java.lang.String
bioProcessId_Attribute	java.lang.String
name_Attribute	java.lang.String
pathwayDiagram_Attribute	java.lang.String
taxonId_Attribute	java.lang.String
ProteinSearchCriteria	
geneId_Attribute	java.lang.String
accessionNumber_Attribute	java.lang.String
description_Attribute	java.lang.String
ProteinHomologSearchCriteria	
proteinId_Attribute	java.lang.String
ProtocolSearchCriteria	
name_Attribute	java.lang.String
ProtocolAssociationSearchCriteria	
clinicalTrialProtocolId_Attribute	java.lang.String
protocolId_Attribute	java.lang.String
ReadSequenceSearchCriteria	
CloneId_Attribute	java.lang.String
GeneId_Attribute	java.lang.String
ReadSequenceId_Attribute	java.lang.String
proteinID_Attribute	java.lang.String
refGeneId_Attribute	java.lang.String
tracefileID_Attribute	java.lang.String
RelationshipSearchCriteria	
RelationshipChildId_Attribute	java.lang.String
RelationshipParentId_Attribute	java.lang.String
relationshipType_Attribute	java.lang.String
SAGEExperimentSearchCriteria	
gene_Attribute	java.lang.String
geneId_Attribute	java.lang.String
expressionFeatureId_Attribute	java.lang.String
organ_attribute	java.lang.String
proteinId_Attribute	java.lang.String
taxonId_attribute	java.lang.String
threshold_Attribute	java.lang.String
type_Attribute	java.lang.String

SequenceSearchCriteria

accessionNumber_Attribute	java.lang.String
cloneId_Attribute	java.lang.String
expressionMeasurementId_Attribute	java.lang.String
geneId_Attribute	java.lang.String
refGeneId_Attribute	java.lang.String
isRefSeq_Attribute	java.lang.Boolean
returnDNA_Attribute	java.lang.String
sequenceType_Attribute	java.lang.String

SNPSearchCriteria

geneId_Attribute	java.lang.String

TargetSearchCriteria

agentId_Attribute	java.lang.String
anomalyId_Attribute	java.lang.String
anomalyDescription_Attribute	java.lang.String
geneId_Attribute	java.lang.String
cancerType_Attribute	java.lang.String
conceptID_Attribute	java.lang.String

TaxonSearchCriteria

abbreviation_Attribute	java.lang.String
chromosomeId_Attribute	java.lang.String
isPreferred_Attribute	java.lang.String
scientificName_Attribute	java.lang.String

TissueSearchCriteria

libraryId_Attribute	java.lang.String

TraceFileSearchCriteria

nameId_Attribute	java.lang.String
cloneId_Attribute	java.lang.String
snpId_Attribute	java.lang.String

2 ENTERPRISE VOCABULARY SERVICES

2.1 Introduction to the NCI Enterprise Vocabulary Services

The NCI Enterprise Vocabulary Services (EVS) were developed in response to the need for consistent shared vocabularies among the various projects and initiatives at the National Cancer Institute. Data systems at NCI grew over time, each designed to meet a specific need, and the vocabularies used for keywords and coding were often inconsistent. These discrepant terminologies and usage led to significant search complexity and data-sharing problems. The EVS is addressing these needs for both science management and science research.

The EVS is a set of services and resources that provide NCI with controlled biomedical vocabularies. It serves the science communities by organizing and translating their specific, distinct, but overlapping, terminologies. The infrastructure of the EVS has both a technical component and a human resource component. The technology includes server hardware; database, editing, and management software; and applications programming interfaces. The human resources include operations staff to look after the servers, vocabulary databases, and software; curators to maintain and update the vocabulary database content; and applications support staff who interface the EVS to other NCI systems.

The two major vocabulary services provided by EVS are: the NCI Thesaurus, a description logic-based NCI-specific biomedical thesaurus; and the NCI Metathesaurus, a collection of over 70 biomedical vocabularies, based on the National Library of Medicine's (NLM) Unified Medical Language System (UMLS) Metathesaurus. Each service has a Java API, as well as Java servlets running on the web server to support vocabulary search and browsing

2.1.1 NCI Thesaurus

The NCI Thesaurus was specifically designed to support the terminologies unique to cancer research and is updated on a monthly basis by the EVS team to reflect the most recent progress in the field. It is tailored to meet the needs of the NCI database systems, with an emphasis on the development of logical and consistent conceptual models. The NCI Thesaurus is intended to provide ease of navigation among its concepts and accurate search and retrieval of data from its databases.

The NCI Thesaurus contains about 18,000 concepts, represented by about 80,000 terms; its primary usage is in the definition and regularization of terms used by NCI in automated systems for keywording, database coding, and information retrieval. The Thesaurus is organized into 17 hierarchical trees covering areas such as Neoplasms, Drugs, Anatomy, Genes, Proteins, Techniques, and others, and is available in two forms.

As a stand-alone vocabulary, the NCI Thesaurus is optimized for database coding, search, and data-mining, and is intended for developers of applications that need a tightly controlled vocabulary. Alternatively, much of the NCI Thesaurus content can be accessed as one of the many vocabularies available through the NCI Metathesaurus.

2.1.2 NCICB Research Initiatives

The National Cancer Institute's emphasis today is on advancing the promise of molecular medicine and the translation of basic science findings into improved treatment and prevention strategies. The insights gained from bioinformatics research and genomic studies, as well as from animal models of human cancer, have the potential to greatly enhance our clinical treatment strategies, but only if the supporting infrastructure to bridge these domains is present. It is imperative that knowledge generated in one domain is accessible in other domains, and that efforts to build infrastructure can effectively serve in multiple arenas.

caCORE provides the Institute with a cancer informatics infrastructure backbone, and a critical dimension of this infrasturucture is the vocabularies which allow various applications to share terminologies and data. Two of the most recently developed vocabularies at NCI are also described in this chapter: the Mouse Models of Human Cancers Consortium (MMHCC) vocabulary and the Core Terminology Reference Model (CTRM) vocabulary.

The NCI Mouse Models of Human Cancers Consortium (MMHCC) is a collaborative effort to generate resources, information, and innovative approaches to the application of mouse models to cancer research. One of its primary goals is to facilitate the exchange of scientific knowledge within the mouse models research community. Towards this end, the MMHCC is assembling a vocabulary to support the comparison and differential analysis of mouse and human cancers.

The Core Terminology and Reference Model (CTRM) was developed to model and deploy vocabulary in certain key domains for use by NCICB applications in a short time frame, through a jump-start approach. The three terminology domains covered by CTRM include anatomy, diseases, and therapeutic agents.

Currently, the MMHCC and CTRM terminologies are defined as stand-alone vocabularies. These two vocabularies, MMHCC and CTRM, have recently been merged in a third namespace, and we anticipate that the NCI Thesaurus will contain the CTRM, MMHCC, and NCI vocabularies in a single namespace by early FY 2003.

2.1.3 NCI Metathesaurus

The NCI Metathesaurus is a *collection* of vocabularies and, in addition to the NCI Thesaurus, includes over 70 biomedical vocabularies. These vocabularies include many of the sources from the UMLS Metathesaurus, such as MeSH, Read (NHS), ICD, and other standard controlled vocabularies, along with additional products licensed by EVS for NCI use, such as Stedman's Medical Dictionary, SNOMED, and MedDRA.

The NCI Metathesaurus maps each term in a given vocabulary to the corresponding terms in all of the other vocabularies, thus enabling users to rapidly gain access to terms in unfamiliar contexts. The Metathesaurus serves as a core infrastructure component at NCI, providing abundant synonymy, English language definitions, and mappings between NCI terminology and external sources. In addition, the NCI Metathesaurus facilitates the identification of gaps and discrepancies in the prevailing terminologies and provides a central maintenance point and a single vocabulary resource. The NCI Metathesaurus is also particularly useful for indexing web documents, and can provide definitions and other semantic information about concepts encountered in NCI web pages.

In its entirety, the Metathesaurus contains more than 800,000 concepts representing over 2 million individual terms. EVS has designed the NCI Metathesaurus for use as both an on-line reference tool as well as a network resource for interactive software and web applications. The specific intent of the Metathesaurus design is to facilitate collaboration, data sharing, and data pooling for clinical trials and scientific databases. NCI Metathesaurus updates are released quarterly on the NCI Metathesaurus server.

In addition to the NCI Thesaurus, vocabularies of particular note that are contained in the Metathesaurus include:

- MedDRA (the Medical Dictionary for Regulatory Activities) is maintained and distributed by TRW Inc. under a contract agreement with the International Federation of Pharmaceutical Manufacturers (IFPMA) acting as trustees for the International Conference for Harmonization (ICH). MedDRA is the international standard terminology for reporting in the pharmaceutical industry. MedDRA is used by NCI to report adverse events.
- ICD-O (the International Classification of Disease - Oncology) is the international standard terminology for reporting cancer incidence and other epidemiologies. It is used at NCI for epidemiology and, in particular, in the SEER (Surveillance, Epidemiology, and End Results) program, an authoritative source of information on cancer incidence and survival in the United States.
- MDBCAC (the Mitelman Database of Chromosome Aberrations in Cancer) contains information culled from the literature by Felix Mitelman, Bertil Johansson, and Fredrik Mertens, and relates chromosomal aberrations to tumor characteristics. The MDBCAC vocabulary provides the terminologies used to code tissues in the chromosome aberration database and is incorporated into several NCICB applications as well.
- PDQ (Physician Data Query) – In addition to the version currently used with the PDQ production website, the NCI Office of Communications has heavily reworked the disease and drug terminologies for NCI. The production PDQ source is identified as a subsource in the Metathesaurus, and the new PDQ is incorporated into both the NCI Thesaurus and Metathesaurus.
- The Lash Tissue Classifications taxonomy was developed by NCBI and is used to code tissues and morphology of samples contained in NCBI and NCI databases, notably the Cancer Genome Anatomy Project.

The NCI Metathesaurus provides matching of like terms along several dimensions. In addition to lexicographic and "sound-alike" criteria, semantic content matching — perhaps the most important measure of similarity — is also available. Specifically, the NCI Metathesaurus's *concept*-based vocabulary allows terms such as APAP, Acetaminophen, Tylenol, Acetamidophenol, etc. to all be linked to the same core concept by a unique identifier.

EVS can also apply concept-based indexing to documents to match terms across the many vocabularies stored with the NCI Metathesaurus. Entire documents containing ASCII text, HTML, or XML-encoded data can be conceptually indexed. Due to the rich synonymy contained in the Metathesaurus, the resulting indexes do not reflect the lexical content of the documents but, instead, their semantic content.

In summary, the NCI Thesaurus contains the scientific, clinical, and administrative concepts that NCI deals with in conducting its operations. The NCI Metathesaurus system maps the NCI Thesaurus — as well as many other biomedical vocabularies — to a set of concepts spanning both basic and clinical research and treatment.

The NCI Thesaurus is continually updated. Since 1999, contractors have created content and edited the NCI EVS, with regular internal and external reviews by the user communities. Each month, new NCI Thesaurus releases are published for NCI databases and other systems to use and are introduced into the NCI Metathesaurus. These smaller monthly releases of the NCI Metathesaurus help update rapidly changing areas in biomedical terminology, such as cancer genetics and cell and molecular biology. Once a year, the NCI issues a major release of the NCI Metathesaurus, thus keeping it updated with the latest releases of the NLM UMLS Metathesaurus.

The remainder of this section of the caCORE User Manual is organized as follows. Section 2.2 describes the contents of three important EVS terminologies: the NCI Thesaurus, the MMHCC vocabulary, and the CTRM vocabulary. Section 2.3 describes the interactive web interface to the NCI Metathesaurus, EVS's Metaphrase server. Section 2.4 outlines the caBIO Java API to the NCI Metathesaurus for application developers. Finally, Section 2.5 describes the format of the downloadable EVS flat files.

2.2 Local NCI Vocabularies

2.2.1 The NCI Thesaurus Vocabulary

The NCI Thesaurus is the reference biomedical vocabulary for the Institute. Built explicitly to meet NCI's needs, the Thesaurus contains many of the codes, keywords, and special purpose vocabulary that are used there. More generally, however, the NCI Thesaurus may become a reference terminology for the larger cancer research community. While it has historically been available only to NCI researchers, as of the release of caCORE 1.0 it will be made freely available to the general public.

The NCI Thesaurus embodies the following features:

- It facilitates Institute-wide consistency in the codification of data;
- The vocabulary is concept-based rather than term-based, thus providing the function of a thesaurus, with concepts organized by meaning;
- The vocabulary includes concepts and terms usable by a broad range of users, including lay clients, researchers, policy-makers, health care professionals, etc.
- It provides approximate matching (using wildcards), thus allowing the user to locate concepts using imprecise terms and, subsequently, to select from among the "best matches."

The need to address the requirements of the general public, researchers, policy-makers, and health care professionals places strong and sometimes conflicting demands on the design of the NCI Thesaurus. On the one hand, there is the demand for consistency and accuracy. For example, in order for coding and retrieval to be accurate, the NCI Thesaurus must be precise, be logically correct, and have a dependable hierarchical structure.

On the other hand, there is the need to provide an easily understood structure — which often leads to overlapping categories and tangled hierarchies of relations. Coupled with the need to serve a broad community and to tolerate variant spellings and imprecise keywording, the challenges are significant.

To address these challenges, NCI has teamed with Apelon, Inc., a software company specializing in the development of biomedical vocabularies. The Metaphrase application (which includes the NCI Thesaurus as *NCI Source*) takes care of some of these needs, and the description logic NCI Thesaurus handles others.

The NCI Thesaurus implementation is based on the Ontylog implementation of description logic (DL) — a computational paradigm that has evolved out of first order algebra as it relates to acyclic directed graph structures and Aristotelian hierarchy.

In the Ontylog DL approach, for purposes of computation, *concepts* correspond to nodes in an acyclic graph, and edges correspond to *roles*. Roles are uni-directional binary relations that hold between concepts. Syntactically, two concepts connected by an edge form a triple. These triples can be thought of as transitive sentences, with the concept at the origin of the edge being the subject, the role being the transitive verb, and the concept pointed to by the edge being the direct object.

Primitive concepts in an Ontylog DL are those atomic elements of the representation that are not defined in terms of any other elements. While they may have a common-sense interpretation to the user, their definitions exist outside the world circumscribed by the vocabulary. Their relations to other non-primitive and primitive concepts define non-primitive concepts. In Ontylog description logic, most concepts obtain their meaning from the role relationships that relate them to other concepts.

The NCI Thesaurus is edited and maintained in the Terminology Development Environment (TDE) provided by Apelon. The TDE is an XML-based system that implements the computational model of description logic based on Apelon's Ontylog Data Model. The Ontylog Data Model uses four fundamental components: *Concepts*, *Superconcepts*, *Kinds,* and *Roles*.

A *concept* is the basic unit of information. An example of a concept is "Breast Ductal Carcinoma." A s*uperconcept* is the concept's parent in the *is_a* hierarchy. For example, the *superconcept* of "Breast Ductal Carcinoma" is "Malignant Breast Neoplasm." In other words, "Breast Ductal Carcinoma" has an *is_a* relationship (is a specialization of) "Malignant Breast Neoplasm."

A *kind* is a major category or subdivision in the NCI Thesaurus. For example, both of the categories "Findings and Disorders" and "Gene" are *kinds*. Each *concept* has a unique *kind*. Other examples of *kinds* include: Anatomic Structures and Systems, Biological Processes, and Clinical or Research Activity. Kinds are used by NCI ro restrict the concepts that may be associated by certain roles. From the user's point of view, the effect of these restrictions is to increase the precision of the NCI Thesaurus.

A *role* is a relationship between concepts. It connects concept to concept and is passed from parent to child in the inheritance hierarchy. For example, a "Malignant Breast Neoplasm" has the role *located-in*, connecting it to the concept "Breast". Thus, since the concept "Breast Ductal Carcinoma" *is-a* "Malignant Breast Neoplasm," it inherits the *located_in* relation to the "Breast" concept.

Another example is *associated-with*, as in a particular gene is *associated-with* a particular disease. These lateral relations among concepts are referred to as associative or semantic roles — in contrast to the hierarchical relations that reflect the *is-a* roles.

In the first order algebra upon which Ontylog DL is based, for every defined relationship, there is also an inverse relation. For example, if *A* is contained by *B*, then *B* contains *A*. Inverse relationships are useful and are expected by human users of ontologies. However they have a computational cost. Classification of the acyclic graph is a computational process used to enforce inheritance and other formal properties of the DL. If the edges connecting concept nodes are bi-directional, then the computation becomes NP hard. Therefore in the Ontolog implementation of DL the edges are uni-directional. Inverse relationships do not exist in the NCI Thesaurus, but they are available to users because they are computed by an application available via the API.

For every defined relationship, there is also an inverse relation. For example, if *A* is contained by *B*, then *B* contains *A*. For simplicity, these relations are labeled as *Xxx* and *inverse-Xxx,* for example*, is-a* and *inverse-is-a*. Currently, inverse relations must be explicitly computed by an application via the API. In future releases they will be computed automatically in the Terminology Development Environment (TDE), and available directly.

Figure 2.2.1 gives a high-level overview of how the NCI Thesaurus is deployed, on both the back- and front-ends, to provide rapid response time to routine maintenance tasks, browser requests, and the Java application programming interface.

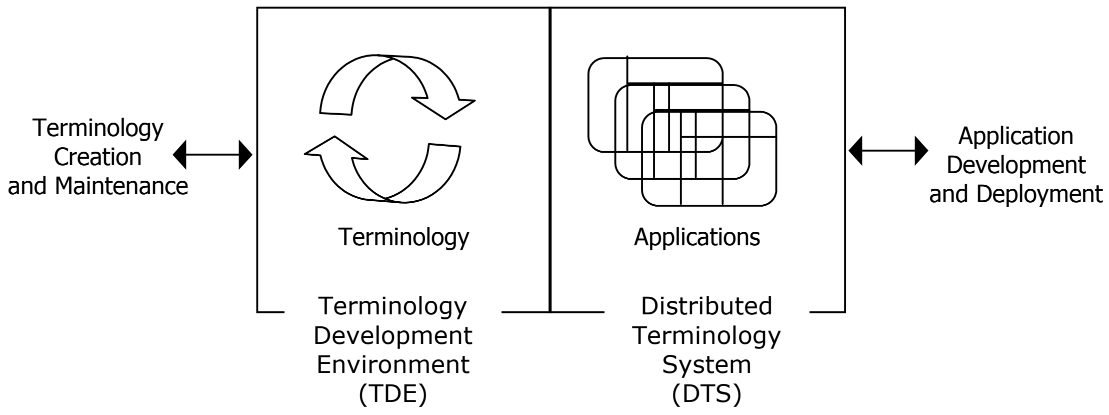

Terminology Creation and Maintenance
Terminology
Applications
Application Development and Deployment
Terminology Development Environment (TDE)
Distributed Terminology System (DTS)

Figure 2.2.1. An overview of the NCI Thesaurus infrastructure

The Distributed Terminology System (DTS) is supplied by Apelon and provides a Java API for programmatic access to the NCI Thesaurus. As this is proprietary software, only a limited interface through the caBIO application's API is available to the public in release 1.0 of caCORE. This interface was described in the preceding section. The contents of the NCI Thesaurus are also available however, as a downloadable flat file, as described in Section 2.5.

The NCI Thesaurus has about 18,000 concepts, represented by about 80,000 terms. These concepts are organized into hierarchies up to 15 levels deep, and are linked by about 85,000 semantic relationships. The NCI Thesaurus is optimized to support NCI's software systems, which are used for database coding and key wording, data-mining, database searching, and text indexing.

Alternatively, much of the NCI Thesaurus content can be accessed as an integral part of the NCI Metathesaurus. The NCI Metathesaurus is a much larger database of terminology, containing over 850,000 concepts represented by about 2 million terms and over 4 million relationships. However, the relationships encoded in the Metathesaurus are much more primitive, as the primary intent of that project is to provide term matching across vocabularies. The NCI Thesaurus is included within the NCI Metathesaurus so that NCI-specific terms can be mapped to analogous terms in standard biomedical vocabularies.

The NCI Thesaurus was created to meet the needs of the NCI database systems and to ensure that concepts would be modeled correctly. Correct modeling of NCI concepts permits reliable navigation and accurate explosion and aggregation of concepts from the NCI databases. For example, the superconcept *breast cancer* can be "exploded" to retrieve all of its subtypes, such as *ductal carcinoma in situ of breast*, *lobular carcinoma of breast*, etc. Conversely, these more specific concepts can be aggregated or "rolled up"

The NCI Thesaurus is continually updated. Subject matter experts create the content of the NCI EVS vocabulary products, with reviews by NCI staff and outside reviewers. Each month, new NCI Thesaurus releases are published. Once a year, there is a major release of the NCI Metathesaurus, which keeps it updated with the latest release of the NLM UMLS Metathesaurus.

2.2.2 The MMHCC Vocabulary

The NCI Mouse Models of Human Cancers Consortium (MMHCC) is a collaborative effort to generate resources, information, and innovative approaches to the application of mouse models to cancer research. One of its primary goals is to facilitate the exchange of scientific knowledge within the mouse models research community. Towards this end, the MMHCC is assembling a vocabulary to support the comparison and differential analysis of mouse and human cancers.

Animal models that can mimic the origin, development, and clinical course of human malignancies provide critical insight to all types of cancer research, including basic, translational, clinical, and epidemiological studies. These models are generated by either transferring new genes into, or inactivating genes already present in, an animal's genome. The animals are then susceptible to certain cancers by the same genetic and environmental conditions that act on humans.

The resulting models are used to study the biology of tumor development and to evaluate new methods of detection, diagnosis, prevention, and treatment. Mice are especially useful, as their genetics, cancer susceptibility, and tumor formation often closely resemble those of humans. In addition, because their tumors develop over months rather than years, the deployment of these models affords rapid advances in our understanding of tumor formation and progression.

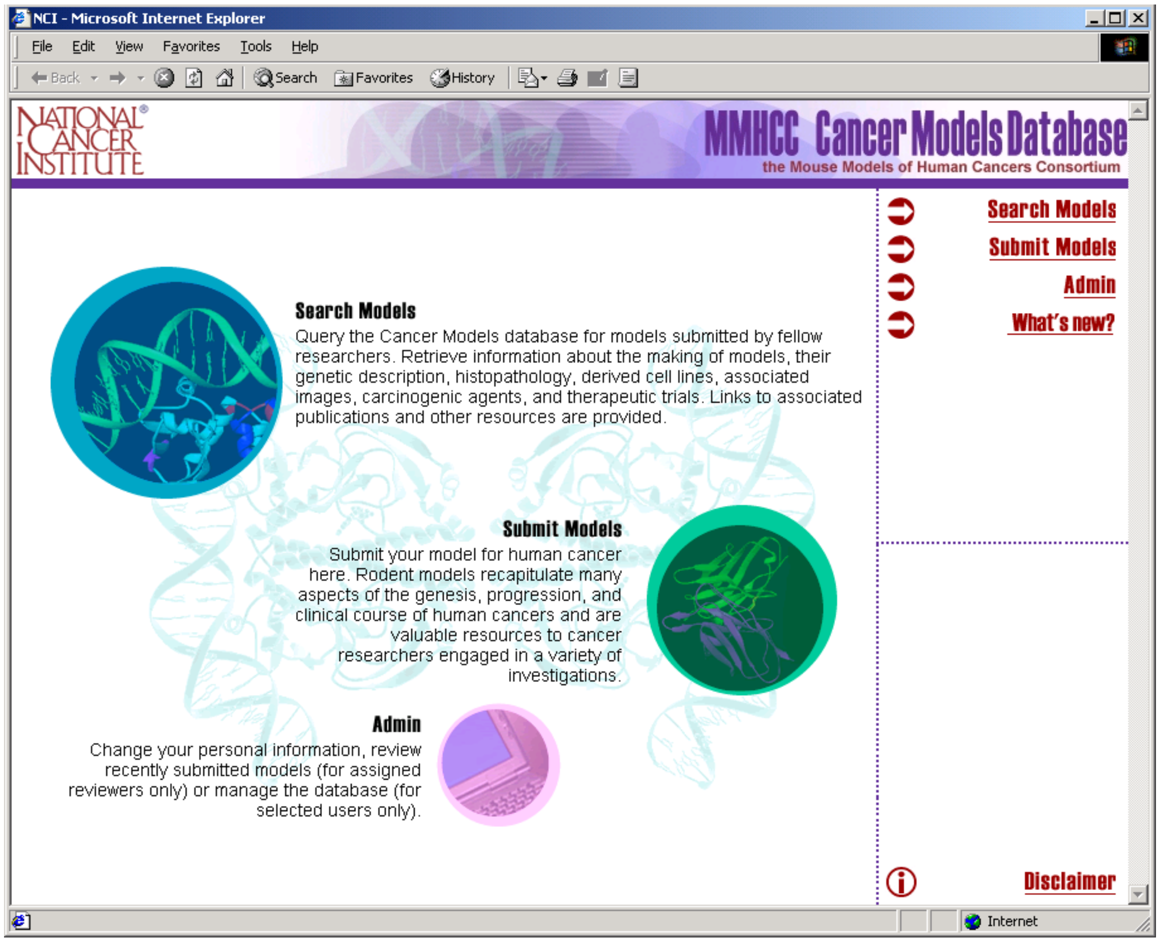
Figure 2.2.2 shows the MMHCC's Cancer Models Database (CMD) website, which allows users to browse the database, submit new models, edit previously submitted models, and manage their personal user accounts. In addition to more general information about the model, each retrieved model page provides access to information about the model's genetic description, carcinogenic interventions, histopathology, therapeutic approaches, cell lines, images, and publications. The general information page specifies the species, strain, and phenotype, along with the name of the investigator and the laboratory where the model was developed.

An important component of the CMD architecture is its EVS-powered interfaces. The EVS project has developed a controlled vocabulary explicitly for use in the study of mouse models of human cancers in conjunction with the MMHCC pathologists, and the CMD organizes the information provided on its search and submission forms around the EVS vocabulary trees.

The EVS MMHCC vocabulary includes disease name, organ/tissue, and strain terminologies. Currently, the anatomical terms cover nine organ systems affected by cancers: breast/mammary gland, prostate, lung/pulmonary, ovary, skin, blood/lymph, brain/neurologic, colon/intestinal tract, and cervical tissues. The disease name terminologies are organized under the following categories:

- Diseases of the Mouse Intestinal Tract
- Mouse Hematologic Disorders
- Diseases of the Human Breast
- Diseases of the Mouse Mammary Gland
- Mouse Nervous System Disorders
- WHO Classification of Human Ovarian Tumors

- Primary Pulmonary Tumors of the Mouse
- Diseases of the Mouse Prostate Gland
- Mouse Skin Disorders


NCI - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
NATIONAL CANCER INSTITUTE
MMHCC Cancer Models Database
the Mouse Models of Human Cancers Consortium
Search Models
Submit Models
Admin
What's new?
Disclaimer
Internet

Search Models
Query the Cancer Models database for models submitted by fellow researchers. Retrieve information about the making of models, their genetic description, histopathology, derived cell lines, associated images, carcinogenic agents, and therapeutic trials. Links to associated publications and other resources are provided.

Submit Models
Submit your model for human cancer here. Rodent models recapitulate many aspects of the genesis, progression, and clinical course of human cancers and are valuable resources to cancer researchers engaged in a variety of investigations.

Admin
Change your personal information, review recently submitted models (for assigned reviewers only) or manage the database (for selected users only).

Figure 2.2.2. The MMHCC Cancer Models Database.

The MMHCC views the deployment of a naming system around murine neoplastic diseases as an important undertaking, as the terminology used in describing human disease processes may not accurately represent the disease processes evident in mice. For example:

- In humans, the notion of "acute" is linked to the percentage of immature cells in the bone marrow *and* the clinical course of the disease, as these are closely correlated. In mice however, the percentage of immature cells in the bone marrow is not closely correlated with clinical course and, accordingly, the term "acute" is not used, as it might convey misinformation.
- In humans, the modifier "myeloid" has multiple possible meanings depending on its context, and refers to an entire group of cell types. In mice, myeloid refers only to granulocytes and monocytes. Thus, to reduce confusion, the murine terminology uses the term non-lymphoid — to avoid confusion as well as to emphasize the differentiation of lymphoid versus non-lymphoid disorders.

These are but two of the many often subtle differences in terminology between the two species.

Most recently, the CTRM and MMHCC vocabularies have been merged into a single namespace, and a vocabulary browser can be used to view this combined vocabulary at http://mmr.afs.apelon.com/CTRM/tree-menu.

Like the NCI Thesaurus, the MMHCC vocabulary uses description logic to represent semantic relations between concepts. Description logics are a family of knowledge representation formalisms rooted in earlier semantic network and frame-based systems, and are based on the notion of *concepts* (classes) and *roles* (binary relations between concepts). (See section 2.2.1 for further discussion.)

In addition to providing understandable, reproducible, and useful terminologies, a concommitant goal of the EVS MMHCC vocabulary project is to establish processes, both manual and electronic, whereby such controlled terminologies can be created and consistently maintained with the help of outside subject experts.

In summary, the goals in creating the EVS MMHCC hierarchies have included:

- Formalizing a comprehensive, consistent naming system for mouse neoplasms, pre-neoplastic disorders, and related diseases as well as making available other relevant vocabulary such as murine strains and histopathologic staining methods;
- Providing a structured framework for use in the cataloguing and retrieval of MMHCC-related resources such as images, dissection protocols, and mouse models;
- Establishing processes, both manual and electronic, whereby such controlled terminologies can be created and consistently maintained with the help of outside subject experts, and
- Providing a structured framework for the comparative anatomy and diagnosis between mice and humans, thereby facilitating validation or certification of mouse models of human disease.

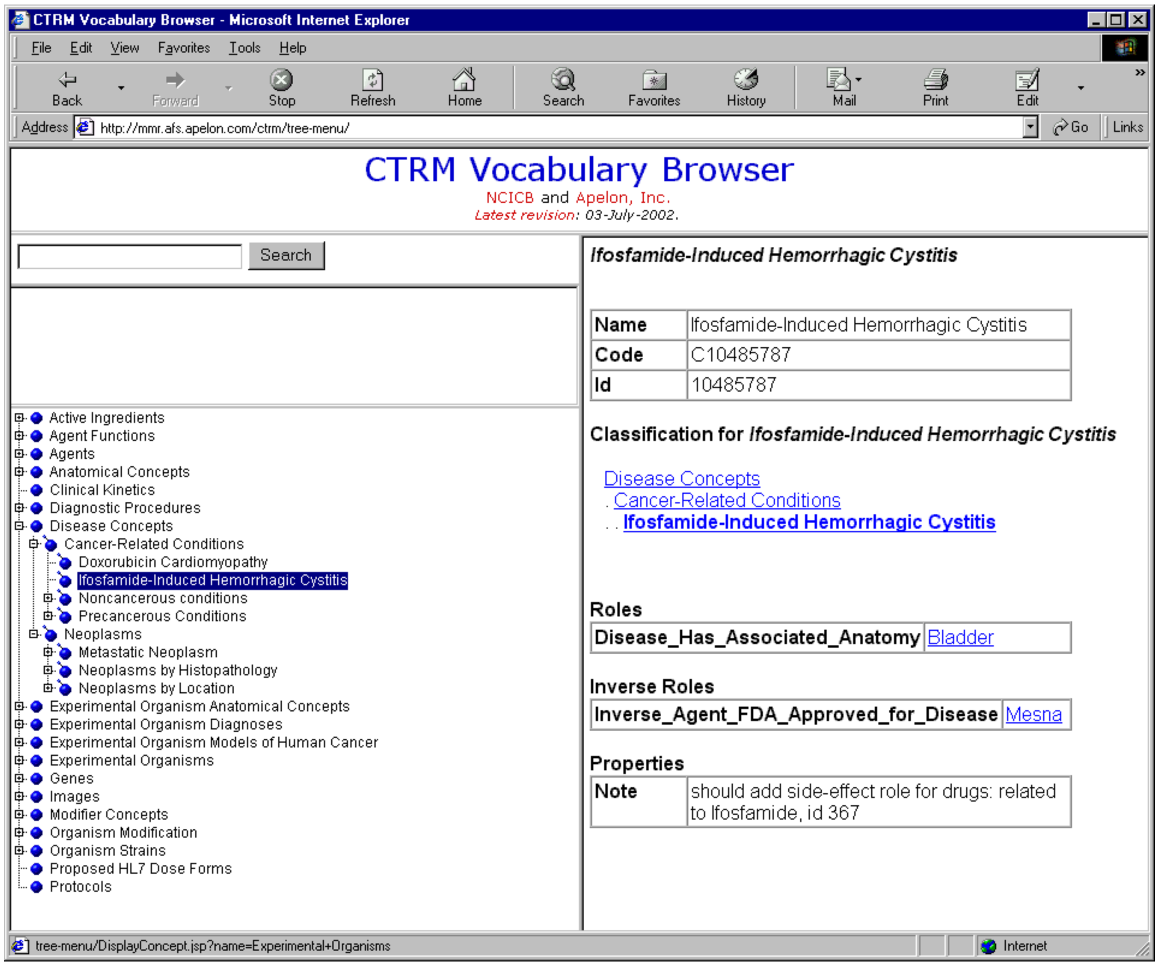
A primary resource for additional information about mouse models is the *emice* website, which provides links to information generated by the MMHCC as well as other NCI-supported projects.

2.2.3 The Core Terminology and Reference Model Vocabulary

The Core Terminology and Reference Model (CTRM) was developed to model and deploy vocabulary in certain key domains for use by NCICB applications in a short time frame, through a jump-start approach. The three major terminology domains covered by CTRM include anatomy, diseases, and therapeutic agents.

NCICB applications, such as CMAP, CGAP, and the Director's Challenge (DC), are at the cutting edge of cross-disciplinary cancer research. As such, much of the computing and application infrastructure in these projects has been deployed in advance of any comprehensive terminology infrastructures in the core areas whose data they depend upon. These applications soon ran into complications resulting from the need to incorporate data from diverse sources coded with idiosyncratic terminologies. The CTRM vocabulary emerged as a rapid response to the need to unify the various representations and facilitate the processing of heterogeneous data.

The CTRM can be viewed on the web via a vocabulary browser. Figure 2.2.3 shows a view inside the Vocabulary Browser, after the user has selected "Ifosamide-Induced Hemorrhagic Cystitis."


CTRM Vocabulary Browser - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://mmr.afs.apelon.com/ctrm/tree-menu/
Go Links
Search
Internet

CTRM Vocabulary Browser
NCICB and Apelon, Inc.
Latest revision: 03-July-2002.

- Active Ingredients
- Agent Functions
- Agents
- Anatomical Concepts
- Clinical Kinetics
- Diagnostic Procedures
- Disease Concepts
  - Cancer-Related Conditions
    - Doxorubicin Cardiomyopathy
    - Ifosfamide-Induced Hemorrhagic Cystitis
    - Noncancerous conditions
    - Precancerous Conditions
  - Neoplasms
    - Metastatic Neoplasm
    - Neoplasms by Histopathology
    - Neoplasms by Location
- Experimental Organism Anatomical Concepts
- Experimental Organism Diagnoses
- Experimental Organism Models of Human Cancer
- Experimental Organisms
- Genes
- Images
- Modifier Concepts
- Organism Modification
- Organism Strains
- Proposed HL7 Dose Forms
- Protocols

Ifosfamide-Induced Hemorrhagic Cystitis

Name	Ifosfamide-Induced Hemorrhagic Cystitis
Code	C10485787
Id	10485787

Classification for *Ifosfamide-Induced Hemorrhagic Cystitis*

Disease Concepts
. Cancer-Related Conditions
. . **Ifosfamide-Induced Hemorrhagic Cystitis**

Roles

Disease_Has_Associated_Anatomy	Bladder

Inverse Roles

Inverse_Agent_FDA_Approved_for_Disease	Mesna

Properties

Note	should add side-effect role for drugs: related to Ifosfamide, id 367

tree-menu/DisplayConcept.jsp?name=Experimental+Organisms

Figure 2.2.3. The CTRM Vocabulary Browser.

Navigation inside the Vocabulary Browser is simple and intuitive; the left panel displays an expandable terminology tree, and the right panel displays summary information about the currently selected concept.

This summary report displays the Name, Code, Id, Classification, Roles and Properties of the selected concept. The classification field re-iterates the concept's position in the hierarchy, tracing back to the top-level concept for that branch of the tree. The properties associated with a concept include external codes, MeSH definitions, alternate and preferred names, synonyms, semantic type, and source information, and any additional notes that might be available.

The Roles (and Inverse-Roles) of a given concept show that concept's relationships to other concepts. Like the other EVS vocabularies, the CTRM is based on description logic embodying concepts (classes) and roles (binary relations between concepts). Roles are uni-directional in this implementation of description logic, relating one concept to another concept in a specified direction. The inverse-role points back in the other direction.

For example, the "Ifosamide-Induced Hemorrhagic Cystitis" concept has the relation *Disease_Has_Associated_Anatomy* with the "Bladder" concept. Clicking on this concept in the right panel links the user to the report summary for "Bladder," where we find the Inverse-Role: *Inverse_Disease_Has_Associated_Anatomy* leading back to "Ifosamide-Induced Hemorrhagic Cystitis."

The significance of CTRM is twofold. Firstly, CTRM serves as an indispensable bridging technology in support of cutting-edge interdisciplinary cancer research projects ongoing at NCI. Secondly, the CTRM project can be viewed as a successful experiment in rapid prototyping of application-specific terminologies. As such, it provides facilities for rapid modeling of domain-specific vocabularies, for simultaneously deploying a new vocabulary as it is being updated, and for mapping terms from different terminologies. Our experience with CTRM is demonstrating that creating an explicit reference model and subsequently mapping existing terms to this model can advance the integration of applications and reduce application maintenance costs.

Currently, the MMHCC and CTRM vocabularies reside in separate namespaces. The MMHCC vocabulary is used by the Cancer Models Database to retrieve mouse anatomy and diagnosis terms. The CTRM vocabulary, which covers Anatomy, Diseases, and Therapeutic Agents, was designed for use by other NCICB applications.

These two vocabularies, MMHCC and CTRM, have recently been merged in a third namespace, and the applications which access these vocabularies separately are now in the process of updating the appropriate Java classes to access this merged version of the two.

The CTRM vocabulary is also in the process of being merged into the NCI Thesaurus. We anticipate that the NCI Thesaurus will contain the CTRM, MMHCC, and NCI vocabularies in a single namespace by early FY 2003.

2.3 The Metaphrase Web Interface

The NCI Metathesaurus provides NCI applications with a comprehensive source of terminology relevant to NCI's operations in a single, integrated resource. The NCI Metathesaurus is available through the web interface described in this section, as well as through a Java API, which is described in the following section.

The NCI Metathesaurus contains over 70 vocabularies, including: most of the public domain and certain proprietary vocabularies in the National Library of Medicine's Unified Medical Language Thesaurus (UMLS sources); vocabularies developed internally at NCI; and external vocabularies that NCI has licensed.

The local vocabularies developed at NCI are listed in Table 2.3.1. Three of these, the NCI Thesaurus, MHCC, and CTRM vocabularies, were described in the previous section. A limited model of the NCI Thesaurus is accessible via the Metaphrase browser, as the NCI Source. The MHCC and CTRM vocabularies have not yet been incorporated into the NCI Thesaurus, but will be in the near future.

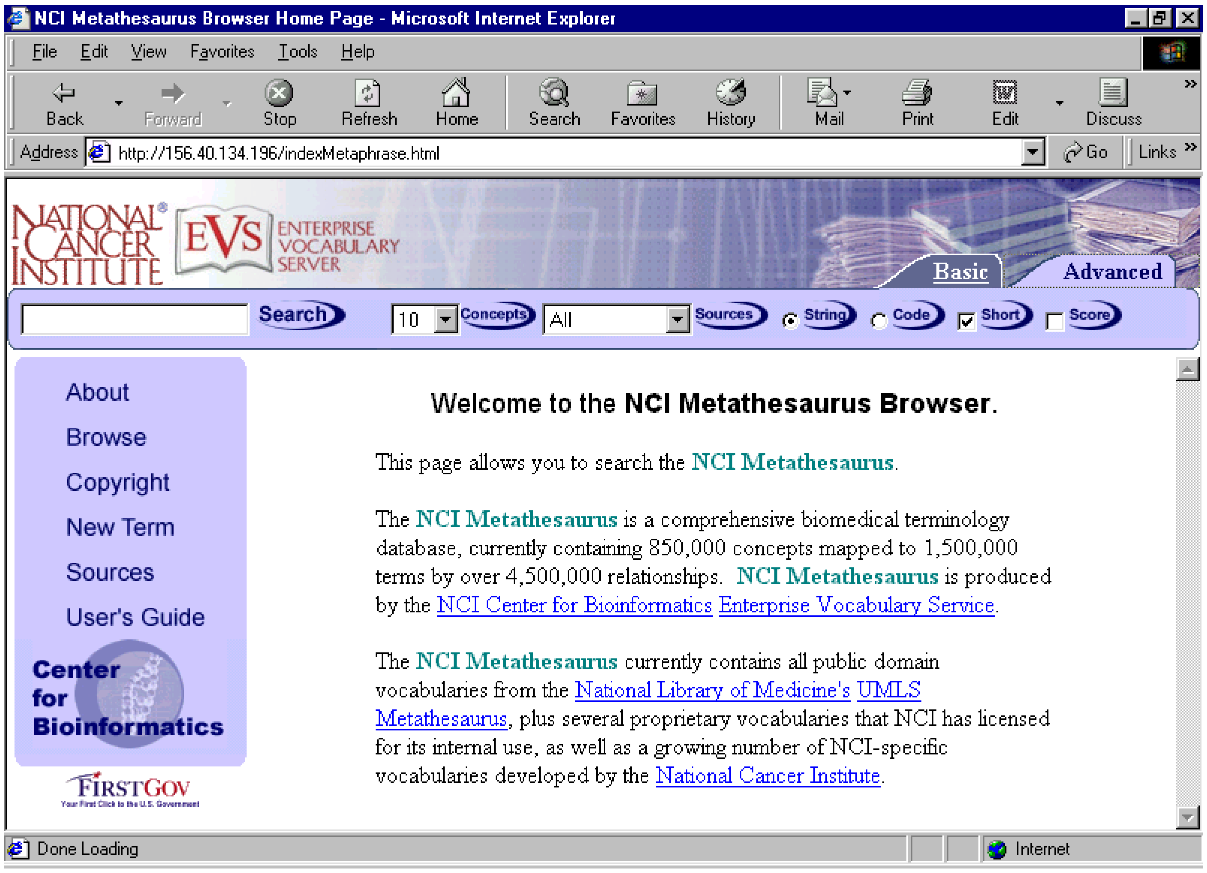
Vocabulary	Content	Usage
NCI Thesaurus	Codes, keywords and special purpose terminology for internal use at NCI	Reference terminology for internal NCI applications
NCIPDQ	Expanded and re-organized PDQ	CancerLit indexing and clinical trials accrual
NCISEER	SEER terminology	Incidence reporting
CTEP	CTEP terminology	Clinical trials administration
MDBCAC	Topology and Morphology	Cancer genome research
ELC2001	NCBI tissue taxonomy	Tissue classification for genetic data such as cDNA libraries.
ICD03	Oncology classifications	Cancer genome research and incidence reporting
MedDRA	Regulatory reporting terminology	Adverse event reporting
MMHCC	Mouse Cancer Database terminology	Mouse Models of Human Cancer Consortium
CTRM	Core anatomy, diagnosis and agent terminology	Translational research by NCICB applications

Table 2.3.1. NCI local source vocabularies included in the Metathesaurus.

The Metaphrase server is a Java application developed by Apelon, Inc. The Metaphrase server implements limited natural language processing and semantic network features, with *lexical matching* applied to locate related terms. Specifically, given a phrase or keyword entered by the user, it searches for definitive terms sharing significant *lexeme*s or roots, with the user's input.

For example, the expression "degenerative joint disease" is said to be *lexically related* to the phrase "Joints, Knee," as they share the "joint" lexeme. Two phrases containing *exactly* the same set of lexemes are said to be *lexically equivalent*, meaning that there is a one-to-one mapping of the significant terms in the two expressions.

Figure 2.3.1 shows the first page of the Metaphrase browser. As indicated by the *basic* and *advanced* folder tabs, the browser provides two levels of interaction. In the basic interface, the user simply enters a keyword or phrase directly into the text search box and presses enter. In the advanced interface, shown in Figure 2.3.1, additional options (described below) are provided to limit the search.


NCI Metathesaurus Browser Home Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://156.40.134.196/indexMetaphrase.html
Go Links
NATIONAL CANCER INSTITUTE
EVS ENTERPRISE VOCABULARY SERVER
Basic
Advanced
Search
10
Concepts
All
Sources
String
Code
Short
Score
About
Browse
Copyright
New Term
Sources
User's Guide
Center for Bioinformatics
FIRSTGOV
Done Loading
Internet

Welcome to the NCI Metathesaurus Browser.

This page allows you to search the NCI Metathesaurus.

The NCI Metathesaurus is a comprehensive biomedical terminology database, currently containing 850,000 concepts mapped to 1,500,000 terms by over 4,500,000 relationships. NCI Metathesaurus is produced by the NCI Center for Bioinformatics Enterprise Vocabulary Service.

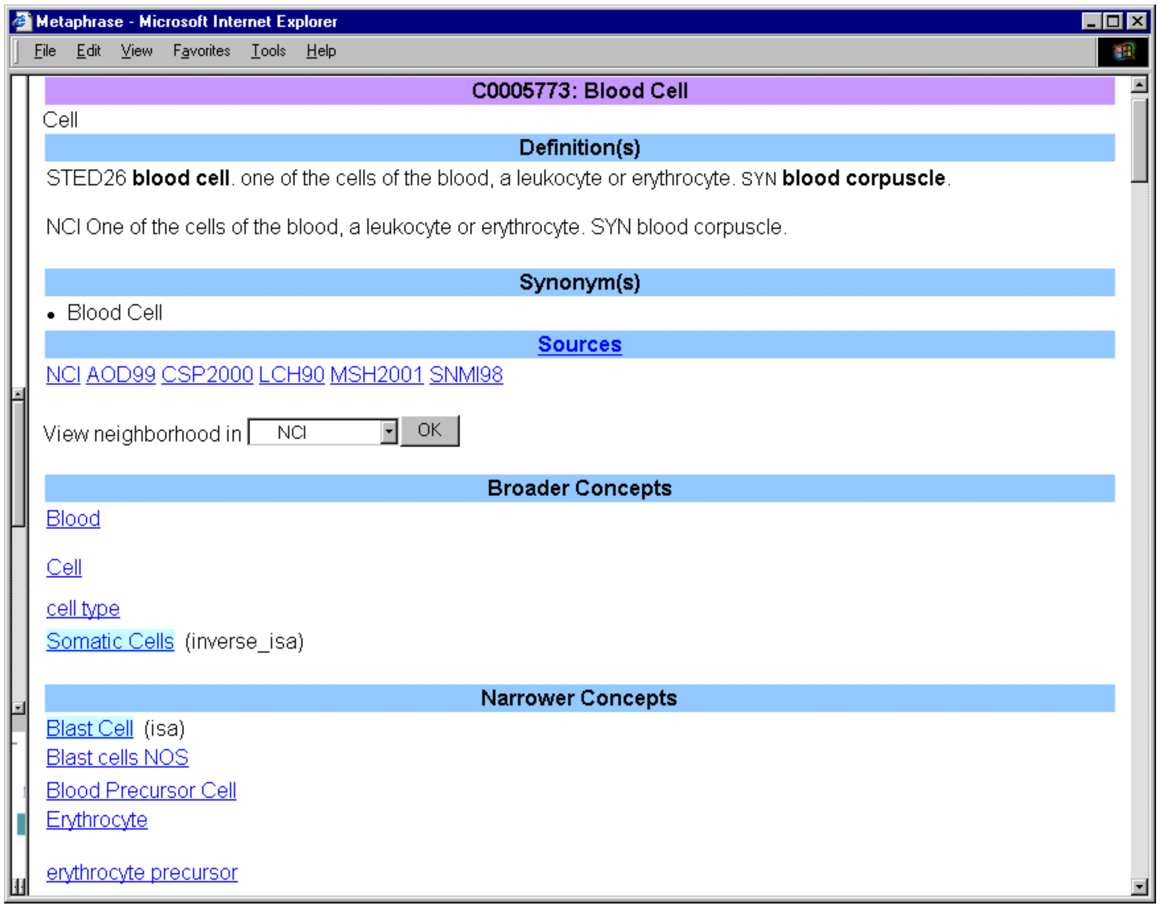
The NCI Metathesaurus currently contains all public domain vocabularies from the National Library of Medicine's UMLS Metathesaurus, plus several proprietary vocabularies that NCI has licensed for its internal use, as well as a growing number of NCI-specific vocabularies developed by the National Cancer Institute.

Figure 2.3.1. The NCI Metaphrase web interface.

In either case, the first response to the user's initial query is a list of matching concepts from the Metathesaurus. The first column in this result list shows the concept's preferred name, and the second column shows its semantic type. For example, the concept "Mouse" has the semantic type "Mammal," while the concept "Knock-Out Mouse" also has the semantic type "Mammal" as well as "Experimental Model of Disease."

Clicking on the (selectable) name of a concept in the left column yields the information page (Figure 2.3.2) for that concept. This page is a compendium of definitions, synonyms, and related concepts, culled from all of the sources whose vocabularies include either the concept itself or a known synonym for the concept. Each definition has a prefix indicating the source providing that definition.

Following the list of synonyms is a list of the sources, with each represented as a selectable hyperlink. Clicking on one of these sources produces a page providing information about the term in that source, including: the term's ID and preferred name [PT] in that vocabulary, synonyms [SY], acronyms [AB], and, if the source is of hierarchical form, the position of that term in the source hierarchy.


Metaphrase - Microsoft Internet Explorer
File Edit View Favorites Tools Help

C0005773: Blood Cell

Cell

Definition(s)

STED26 **blood cell**. one of the cells of the blood, a leukocyte or erythrocyte. SYN **blood corpuscle**.

NCI One of the cells of the blood, a leukocyte or erythrocyte. SYN blood corpuscle.

Synonym(s)

- Blood Cell

Sources

NCI AOD99 CSP2000 LCH90 MSH2001 SNMI98

View neighborhood in NCI OK

Broader Concepts

Blood

Cell

cell type

Somatic Cells (inverse_isa)

Narrower Concepts

Blast Cell (isa)

Blast cells NOS

Blood Precursor Cell

Erythrocyte

erythrocyte precursor

Figure 2.3.2. The Information page for a Concept in the Metaphrase browser.

Underneath the list of sources on the information page is a drop-down menu box entitled "View Neighborhood." Selecting a source in this box and clicking *OK* produces an expanded list of all *semantically* related ("nearby") concepts in that terminology. Not all vocabularies include semantic relations; these specify additional dependencies beyond the simple inheritances implied by the concept hierarchies and convey relationships such as "caused by", "contains", etc. For vocabularies defining semantic networks of such relations, the concepts included in the "neighborhood" are those removed from the current concept by just one link.

2.3.1 Navigating Over Related Concepts

The information page (Figure 2.3.2) also displays additional concepts related to the current selection. These related concepts are broken down into Broader Concepts, Narrower Concepts, and Related Concepts. Not all sources possess hierarchies. Broader and Narrower Concepts are derived from those sources that do contain hierarchy structures. Taken across all sources with hierarchies in which the concept occurs:

- Each concept may have one or more broader concepts whose semantic content is a generalization of the selected concept, and
- A concept may have 0 to many descendants, where each descendant concept is a specialization of the current concept.

Thus, the list of *broader* concepts is the compendium of antecedent concepts from all of the sources that have hierachies, and the list of *narrower* concepts is the set of all descendant concepts over all such vocabularies.

The list of Related Concepts encompasses a broader and less well-defined set of relations, as it depends on the semantic relations defined in the contributing vocabularies. Some vocabularies, such as the NCI Thesaurus, define very sophisticated and specific relations, such as the fact that a particular bacterium is the etiologic agent of a specific disease. Other sources provide only primitive relations indicating that two concepts depend on one another in unspecified ways.

All of the concepts listed as either Broader Concepts, Narrower Concepts, or Related Concepts are hyperlinked to the corresponding information pages for those concepts. Some of these are annotated to indicate the specific relation that is referenced.

For example, the concept “glioblastoma” lists “Common Neoplasm” as a broader concept through an *inverse_isa* relation, meaning that “glioblastoma” *is_a* (type of) “Common Neoplasm.” Similarly, many of the descendant concepts listed as narrower concepts are related to the parent concept through direct *is_a* relations, e.g., “gliosarcoma” *is_a* (type of) “glioblastoma.”

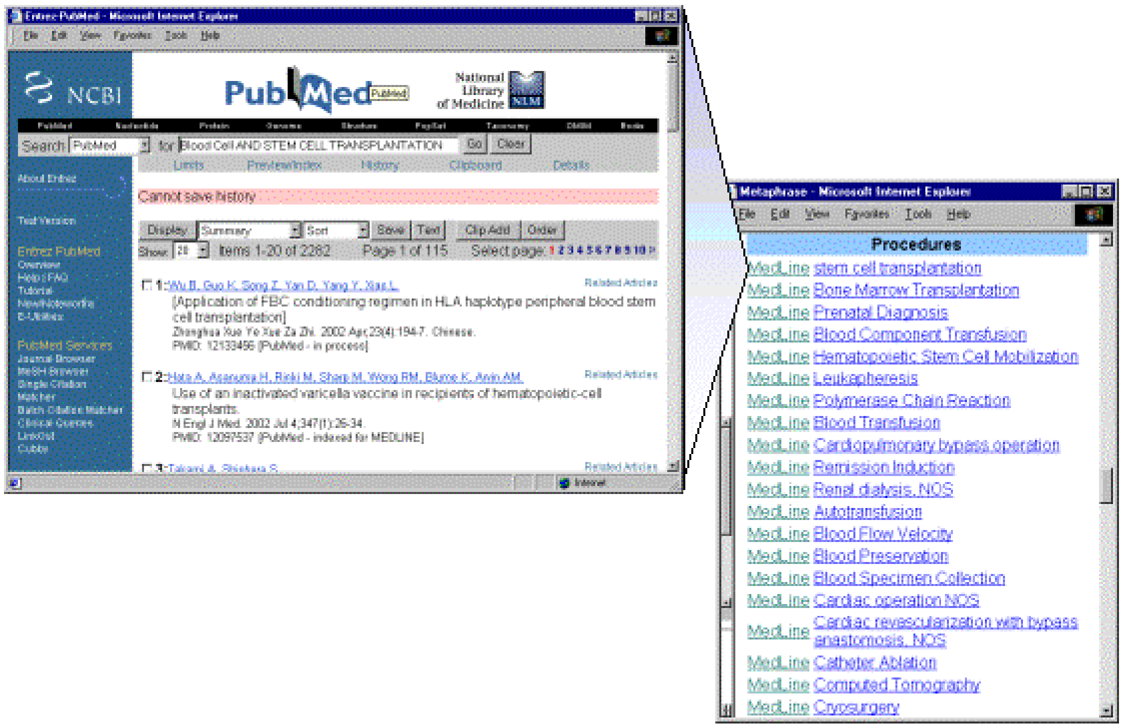
Some of the related concepts for this example are particularly interesting, and include “Astrocytomas,” which is referenced via the *mapped_from* relation; “Brain” and “Central Nervous System,” which are both referenced through the *location_of* relation; and “GLI1 Protein,” which has the relation *Malfunction_Is_Associated_With_Disease*. Finally, those related concepts that are contained in any of the NCI local sources are highlighted in blue.

2.3.2 MeSH Headings Occurring in the Metathesaurus

In addition to the content described thus far, the information pages for concepts that are also MeSH headings provide links to supplemental concepts that co-occur in MedLine with the concept of interest. These are grouped into four categories: Medications, Procedures, Laboratory, and Diagnosis. The most commonly co-occurring concepts appear at the top in each category.

Each supplemental concept is preceded by a MedLine hyperlink, and clicking on that hyperlink opens a new window connected to the NCBI Entrez browser, which provides a list of clinically relevant articles indexed by Medline. Alternatively, clicking on the concept itself brings up the summary information page for that concept provided by Metaphrase.

Figure 2.3.3 for example, shows the procedures associated with the concept “blood cell,” along with the Medline references displayed by the Entrez browser when the link for “stem cell transplantation” is selected.


Entrez-PubMed - Microsoft Internet Explorer
File Edit View Favorites Tools Help
NCBI
PubMed
National Library of Medicine NLM
Search PubMed for Blood Cell AND STEM CELL TRANSPLANTATION Go Clear
Limits Preview/Index History Clipboard Details
About Entrez
Text Version
Entrez PubMed
Overview
Help | FAQ
Tutorial
New/Noteworthy
E-Utilities
PubMed Services
Journal Browser
MeSH Browser
Single Citation Matcher
Batch Citation Matcher
Clinical Queries
LinkOut
Cubby
Cannot save history
Display Summary Sort Save Text Clip Add Order
Show: 20 Items 1-20 of 2282 Page 1 of 115 Select page: 1 2 3 4 5 6 7 8 9 10
Related Articles
Internet
Metaphrase - Microsoft Internet Explorer
File Edit View Favorites Tools Help

1: Wu B, Guo K, Song Z, Yan D, Yang Y, Xia L.
[Application of FBC conditioning regimen in HLA haplotype peripheral blood stem cell transplantation]
Zhonghua Xue Ye Xue Za Zhi. 2002 Apr;23(4):194-7. Chinese.
PMID: 12133456 [PubMed - in process]

2: Hata A, Asanuma H, Rinki M, Sharp M, Wong RM, Blume K, Arvin AM.
Use of an inactivated varicella vaccine in recipients of hematopoietic-cell transplants.
N Engl J Med. 2002 Jul 4;347(1):26-34.
PMID: 12097537 [PubMed - indexed for MEDLINE]

3: Takami A, Shiobara S.

Procedures

MedLine stem cell transplantation
MedLine Bone Marrow Transplantation
MedLine Prenatal Diagnosis
MedLine Blood Component Transfusion
MedLine Hematopoietic Stem Cell Mobilization
MedLine Leukapheresis
MedLine Polymerase Chain Reaction
MedLine Blood Transfusion
MedLine Cardiopulmonary bypass operation
MedLine Remission Induction
MedLine Renal dialysis, NOS
MedLine Autotransfusion
MedLine Blood Flow Velocity
MedLine Blood Preservation
MedLine Blood Specimen Collection
MedLine Cardiac operation NOS
MedLine Cardiac revascularization with bypass anastomosis, NOS
MedLine Catheter Ablation
MedLine Computed Tomography
MedLine Cryosurgery

Figure 2.3.3. Metaphrase hyperlinks (in green) to Entrez PubMed references.

2.3.3 Advanced Browsing Options

As mentioned at the start of this section, Figure 2.3.1 displays the advanced interface for the Metaphrase browser. Thus far, however, we have not referred to any of the options provided by this feature, as our discussion has focused instead on navigating through the result pages. The advanced options are provided to the user as buttons on the main menubar (see Figure 2.3.4 below).

Starting from the leftmost position, a textbox appears, allowing the user to enter keywords or phrases to search for. In the simplest use of this interface, the user merely enters the desired concepts and presses the *Search* button. This corresponds to the Basic interface.

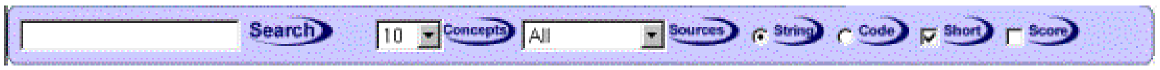


Figure 2.3.4. The advanced options menubar.

The remaining buttons allow the user to:

- Limit the number of matching concepts returned in the results list (*Concepts*);
- Restrict the search to a selected vocabulary (*Sources*);
- Search by string matching or by code (*String, Code*); and
- Limit the number of lexical matches (*Short, Score*).

By default, the maximum number of concepts returned on a single query is 10, and only the highest-quality lexical matches are shown. Unless explicitly reset, the sources used in concept matching will include all of the vocabularies in the Metathesaurus, and the matching will be performed on concept names. In cases where one knows the encoded ID for the concept and the vocabulary in which it is defined, using the *Code* and *Sources* options can speed up the search.

2.3.4 Viewing the NCI Thesaurus

Alternatively, it is possible to bypass the search interface altogether, and simply select concepts one is interested in from the NCI Thesaurus. To do this, select the *Browse* button on the left-hand panel of resources. This will pop up a new browser window displaying the Thesaurus's expandable terminology tree. Clicking on links in this window causes the associated information pages to appear in the Metaphrase browser window. Thus, the NCI Thesaurus effectively serves as an ontology browser for the content of the NCI Metathesaurus.

Figure 2.3.5 shows a screen shot of two views of the NCI Thesaurus terminology tree. The shot on the left shows the entire collapsed top-level vocabulary. Superimposed over that is an expanded view of the top-level *Conceptual Entities* term.

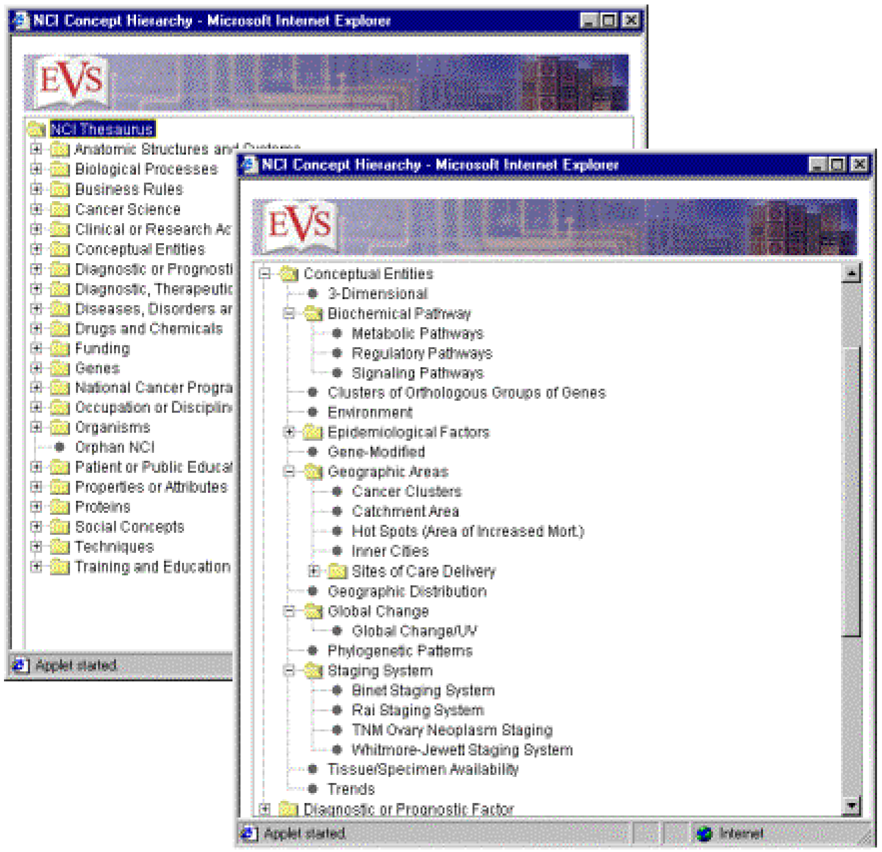


Figure 2.3.5. The NCI Thesaurus vocabulary tree.

The NCI source in the NCI Metathesaurus is based on the NCI Thesaurus, which is an extremely powerful vocabulary resource that uses description logic to capture complex semantic relationships among its terms. The description logic structure of NCI Thesaurus can express sophisticated semantics that cannot be represented in the UMLS-based NCI Metathesaurus. Thus, the NCI source is an approximation of the NCI Thesaurus, but not a full approximation.

2.4 The caBIO Java API to the Enterprise Vocabulary Services

The caBIO project and its applications programming interfaces (APIs) are described in detail in Section 1 of this manual. In addition to descriptive overviews of the caBIO architecture and the data sources to which it provides access, Section 1 gives detailed instructions on how to download the necessary files and set up an environment to use the caBIO APIs.

In this section we describe a Java package within the caBIO infrastructure that was developed as a simple interface to the EVS terminologies. Figure 2.4.1 depicts the five Java classes participating in this interface.

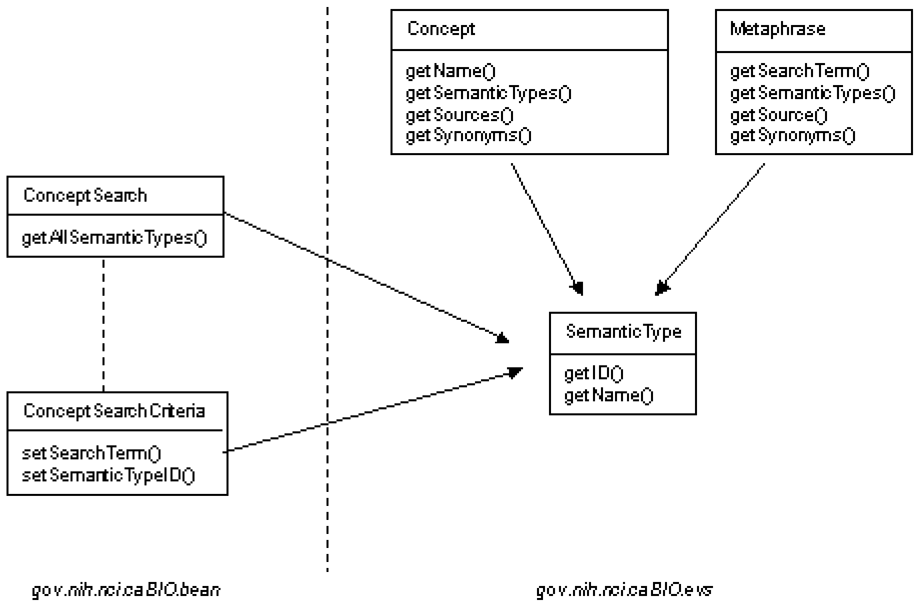


Figure 2.4.1. The caBIO-EVS Java interface.

On the left-hand side of Figure 2.4.1, two classes are shown that are defined in caBIO's *gov.nih.nci.caBIO.bean* package. The three classes on the right are defined in a separate package, named *gov.nih.nci.caBIO.evs*.

The *ConceptSearch* class is considered to be a "domain object" and, as described in Section 1, like all domain objects, has an associated *SearchCriteria* object, which in this case is named *ConceptSearchCriteria*. The search criteria paradigm deployed by caBIO provides powerful search mechanisms, which we summarize briefly here. For a more detailed description of the *SearchCriteria* class, see Section 1.2.

Each *SearchCriteria* object has a group of settable attributes that can be used to define the terms or objects the user wishes to retrieve from the data sources. The domain object's method is then invoked on this *SearchCriteria* object, and the return value is a *SearchResult* object.

In addition to containing an array of the results retrieved from the data sources, a *SearchResult* object contains certain bookkeeping information, such as the indices of the first and last result, the total number of results returned in this batch, and whether or not further results are yet available. In the event that more results are available, the *SearchResult* object can be used to generate a new *SearchCriteria* object with the same attribute values as the original, but with the additional constraint that the first result's index should pick up where the previous *SearchResult* array ended.

Figure 2.4.2 shows a section of Java code that uses some of the objects depicted in Figure 2.4.1 to retrieve concepts related to the keyword "breast."

```
ConceptSearch cs = new ConceptSearch();
ConceptSearchCriteria csc = new ConceptSearchCriteria();
csc.setSearchTerm("breast");
Concept[] concepts = cs.search(csc);

if(concepts!=null){
  for (int i=0; i < concepts.length; i++){
    Concept con = concepts[i];
    System.out.println("Name = " + con.getName());

    SemanticType[] st = concepts[i].getSemanticTypes();
    for (int j = 0; j < st.length; j++){
      System.out.print("Semantic type " + j + ") = ");
      System.out.println(st[j].getName() + " (" + st[j].getID() + ")");
    }

    String[] src = concepts[i].getSources();
    for (int j = 0; j < src.length; j++) {
      System.out.println("Source " +j+") = "+ src[j]);
    }

    String[] syn = concepts[i].getSynonyms();
    for (int j = 0; j < syn.length; j++) {
        System.out.println("Synonym " +j+") = "+ syn[j]);
    }
    System.out.println("\n=================================\n");
  }
}
```

Figure 2.4.2. Using the caBIO EVS Java to retrieve concepts.

The complete specifications for the caBIO packages and class definitions are available on the caBIO project's JavaDoc pages. Two examples of full-scale implementations that deploy the caBIO EVS interface are the CMAP project's website and the Cancer Models Database (CMD), hosted by the MMHCC project. The CMD project uses the caBIO's EVS interface to access the MMHCC vocabulary, which is described in Section 2.4.

Figure 2.4.3 shows the output generated by executing the code in Figure 2.4.2. The program begins by using the *ConceptSearchCriteria* object to define keywords to which terms should be matched. This step corresponds to entering keywords or phrases in the Metaphrase browser's textbox. Next, invoking the *search()* method corresponds to pressing the *Search* button on the browser.

The matching concepts returned by the search are then iterated over in the main loop, where the *Concept* object's methods *getName(), getSemanticTypes(), getSources(),* and *getSynonyms()* are applied, with the results printed to the screen.

```
Name = Breast

Semantic type0) = Body Part, Organ, or Organ Component (T023)

Source 0) = NCI
Source 1) = MDBCAC
Source 2) = NCI

Synonym 0) = Breast

=================================

Name=Breast, NOS

Semantic type0)=Body Part, Organ, or Organ Component (T023)

Source 0)=ICDO3
Source 1)=ICDO3
Source 2)=NCI
Source 3)=ICDO3
Source 4)=AOD99
Source 5)=CCPSS99
Source 6)=CSP2000
Source 7)=LCH90
Source 8)=LNC10o
Source 9)=MSH2001
Source 10)=MTH
Source 11)=RCD99
Source 12)=SNMI98
Source 13)=UWDA142

Synonym 0)=Breast, NOS
Synonym 1)=Mammary gland, NOS
Synonym 2)=Mamma

=================================

Name=Procedures on breast

Semantic type0)=Therapeutic or Preventive Procedure (T061)

Source 0)=CCS99
Source 1)=ICD10AM
Source 2)=MTH
Source 3)=RCD99
Source 4)=SNMI98

Synonym 0)=Procedures on breast
Synonym 1)=Breast
```

Figure 2.4.3. Output generated by executing the code in Figure 2.4.2.

2.5 Downloadable Flat File Formats

The concepts stored in the NCI Thesaurus vocabulary are available for download from the caCORE ftp site. The format of these files is extremely simple, and a good deal of consideration went into making these formats easily parse-able. For each concept, the download file includes the following information:

1. The concept code: all terms have the "C" prefix, followed by its integer index;
2. The preferred concept: this name may contain embedded punctuation and spaces;
3. A list of all parent concepts, as identified in the NCI Thesaurus by the *inverse_isa* relations;
4. A list of synonyms, the first of which is the preferred name;
5. One of the NCI definitions for the term – if one exists.

Each of these separate types of information is tab-delimited; within a given category, the individual entries are separated by pipes ("|"). Only the third and fourth categories, i.e. the parent concepts and synonyms, have multiple entries requiring the pipe separators. Note that while much of the information available from the interactive Metaphrase server is included in the download, any information outside the NCI Thesaurus description logic vocabulary (e.g., Diagnosis, Laboratory, Procedures, etc.) is not.

For example, the flat file download for the term "*Mercaptopurine*" is as follows:

```
C6  Mercaptopurine      Immunosuppressants|Purine Antagonists
    Mercaptopurine|1,3-AZP|1,7-Dihydro-6H-purine-6-thione|3H-Purine-6-thiol|6
Thiohypoxanthine|6 Thiopurine|6-MP|6-Mercaptopurine|6-Mercaptopurine
Monohydrate|6-Purinethiol|6-Thiopurine|6-Thioxopurine|6H-Purine-6-thione,
1,7-dihydro- (9CI)|6MP|7-Mercapto-1,3,4,6-tetrazaindene|AZA|Alti-
Mercaptopurine|Azathiopurine|BW 57-323H|CAS
50442|Flocofil|Ismipur|Leukerin|Leupurin|MP|Mercaleukim|Mercaleukin|Mercap|Me
rcaptina|Mercapto-6-purine|Mercaptopurinum|Mercapurin|Mern|NCI-C04886|NSC
755|Puri-Nethol|Purimethol|Purine-6-thiol (8CI)|Purine-6-thiol
Monohydrate|Purine-6-thiol, Monohydrate|Purinethiol|Purinethol|U-4748|WR-2785
    An anticancer drug that belongs to the family of drugs called
antimetabolites.
```

3 CANCER DATA STANDARDS REPOSITORY

3.1 The NCI Cancer Data Standards Repository

A critical factor in the advancement of translational research on the frontiers of basic and clinical research is the ease with which data can be shared and exchanged. Several obstacles to data sharing in the scientific and medical communities include:

- Different repositories may store the same data but use different data type descriptors in their interfaces and/or documentation, thus obscuring the actual contents;
- In clinical trials, there are no set standards regarding the contents of report forms — either in the naming conventions for the data fields or in the types of values that can populate these fields;
- Various locations may specify different sets of permissible values for the same data types;
- Different units of measures (e.g., months versus years) are applied to the same variables in different contexts.

The immediate consequences of these practices are that *ad hoc* translation filters must be constructed with each new effort to share data, only to be discarded months later as the projects change or come to completion. In the worst case scenario, the incompatibility of the data sets prevents comparative analysis efforts altogether. Longer-term consequences include the lengthy approval cycles entailed for new clinical trials, and the enormous duplication of work involved in all of these efforts.

NCICB supports a broad initiative to standardize the data elements used in clinical trials data capture and reporting. The NCICB Cancer Data Standards Repository (caDSR) is on the cutting edge of new technologies emerging to address these issues. Based on the ISO/IEC 11179 standard for *Data Elements*, the Oracle Service Industries (OSI) division has developed a flexible, customizable registry service with an Oracle 8i database back-end for the controlled curation of terminology to be used in shared clinical and basic research data.

A more detailed description of the ISO/IEC 11179 standard for data elements and how they are to be administered is provided in the next section. Briefly, a data element has both a representational component (its value) and a conceptual component (its semantic interpretation). Each of these components also has a domain of legitimate values or concepts it can assume. The data element in its entirety (i.e. its component and domain specifications) must pass through a registration and approval process before it is posted in the registry. In addition, each element has an assigned *steward* or point of contact.

The core model of the ISO/IEC standard includes provisions for the following components:

- Data elements
- Value domains
- Data element concepts
- Conceptual domains
- Classification schemes

In addition to the provisions specified in the ISO/IEC standard, the caDSR adds the following administered components: *Protocols* and *Questionnaire Content Elements*. Like the core model components, these two extensions require registration and stewardship.

The *Common Data Elements* (CDEs) stored in the caDSR define a comprehensive set of standardized metadata descriptors for cancer research terminology and clinical trials protocols and forms. These data elements have been developed by various NCI-sponsored clinical trials organizations; the data are centrally stored and managed at NCICB in the caDSR.

Programs that have CDE development projects underway include:

- The Cancer Therapy Evaluation Project (CTEP)
- Specialized Programs of Research Excellence (SPOREs)
- Mouse Models of Human Cancers Consortium (MMHCC)
- The Early Detection Research Network (EDRN)
- The Division of Cancer Prevention (DCP)
- The Biomedical Imaging Program (BIP)

Extramural involvement has also included work with the International Collaboration to Screen for Lung Cancer (ICScreen) and the International Association for the Study of Lung Cancer (IASLC) on the collaboration of studies of lung cancer screening with Spiral CT. The caDSR platform provides domain-specific work areas referred to as *contexts* for these various projects accessing the registry. Thus it is possible to maintain coincident non-redundant terminologies with alternative yet unambiguous semantic interpretations that vary with the application.

The NCI caDSR provides a relational database and user interface in support of the workflows involved in the creation, curation, and deployment of CDEs for the advancement of translational research in the treatment and prevention of cancer. This section of the caCORE Manual gives an introduction to the underlying design structure of this registry and an overview on how to use it. The remainder of this chapter is organized as follows.

The next section provides a review of the proposed ISO/IEC 11179 standard for Data Elements. Next, a description of the user interface to the caDSR is given, followed by an overview of the caDSR data model. Section 3.5 provides a catalog of the tables in the caDSR, and Section 3.6 contains the complete data model, including all tables along with their relations and field names. A separate document, The caCORE 1.0 caDSR Database API, provides detailed information on the PL/SQL stored procedures.

3.2 Data Elements in the ISO/IEC 11179 Standard

The ISO/IEC 11179 Specification and standardization and registration of data elements and associated metadata is a draft standard being developed by the JTC1 (Joint Technical Committee 1) Data Management and Interchange Subcommittee (SC3). The purpose of the ISO/IEC 11179 standard is to support the identification, definition, registration, classification, management, standardization, and interchange of data elements, and to promote the sharing and exchange of data throughout the international community. This standard has six parts:

- Part 1: Framework for the specification and standardization of data elements
- Part 2: Classification for data elements
- Part 3: Basic attributes and registry metadata
- Part 4: Rules and guidelines for the formulation of data definitions
- Part 5: Naming and identification principles for data elements
- Part 6: Registration of data elements

The NCI caDSR is an ISO/IEC 11179 compliant database system that was originally build by the Oracle Service Industries division for the U.S. Census Bureau and is also the basis for the caDSR. Oracle's version of this registry is based on the UML metamodels and attribute definitions of the ISO/IEC FDIS 11179-3:2002. It is the intention of the caDSR project to provide a *Conforming* implementation at *Level 2* as described in the standard. Extensions to the standard are administered components *Protocols* and *Questionnaire Content Elements*.

The Data Element Registry provides a reusable, customizable framework for the development of a custom registry. A repository provides only for the storage management of data or metadata. A *registry*, however, additionally provides a registration process for standardization and authority management of the information it maintains. Developing a controlled and registered terminology requires rigorous definitions of both the components that will populate the registry and the protocols that will limit how new terms are entered and persistent terms are maintained. An Oracle development team has worked with NCI to adapt this framework to the needs of the cancer research community, with careful emphasis placed on the need to standardize the terminologies germaine to basic research, patient care, and clinical trials management and protocols.

3.2.1 Concepts and Terminology

The ISO/IEC 11179 standard defines a **Data Element** as a unit of data that in a certain context is considered indivisible. Often the terms "variable," "code," and "field" are used synonymously to mean a Data Element (e.g., Person Name, Person Age, Hospital ID, etc.). As depicted in the UML diagram in Figure 3.2.1, each data element has both a *value* and a *concept* associated with it. The cardinality constraints in Figure 3.2.1 specify that:

- A Data Element has exactly one Data Element Concept and exactly one Value Domain associated with it.
- A Data Element *Concept* has exactly one Conceptual Domain and any number of associated Data Elements.

- A Conceptual Domain may have any number of Data Element Concepts and Value Domains associated with it.
- A Value Domain has exactly one Conceptual Domain and any number of associated Data Elements.

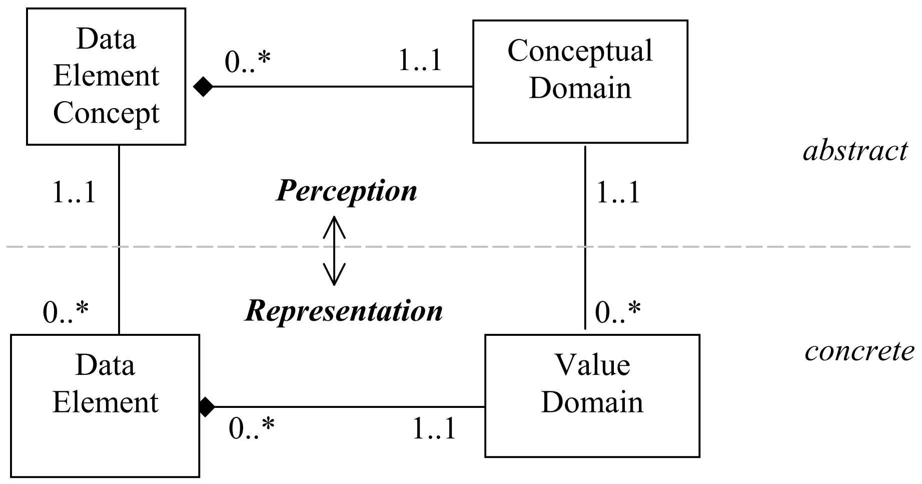


Figure 3.2.1. The basic ISO/IEC 11179 UML Model.

The Data Element's **Value Domain** (VD) defines the set of permissible values for the element, thus constraining the specific value that can be assigned to any particular instance of that Data Element. The Value Domain specifies the data type (e.g., character, number, date), format (e.g., number formats, date formats, ASCII code, Unicode), and, optionally, the unit of measure the values are to be expressed in. For example, the Value Domain for the Data Element *Person Name* might specify that the data type is character and the format is ASCII.

A Value Domain can be enumerated or non-enumerated. For non-enumerated, a range may be specified by its lower and upper bounds, but it will not have any Permissible Values (see below). Value domains can also be related to each other, and the relationship can be specified — for example, part-of, similar-to, etc.

A Permissible Value satisfies the constraints of the element's Value Domain, and has both a concrete value as well as a Value Meaning associated with it. For example, a Permissible Value for the Value Domain "Postal U.S. State" might be "AL," with the Value Meaning "ALABAMA." Value Meanings may be maintained and reused.

While a Data Element's Value has a physical representation (data type, format, etc.), a Data Element Concept does not. A Data Element Concept is used for grouping similar Data Elements, and consists of an **Object Class** and a **Property**. The element's Object Class is an abstraction in the real world that is being modeled (Person, Disease, etc.). An element's Property is a peculiarity common to all members of the element's Object Class. It is much like an "attribute" in relational terms, with the important exception that a Property does not have a specified representation. The Data Element Concept is often named by its Object Class and Property (e.g., "Person Age," "Person Sex"). Data Element Concepts can also be related to each other by relationships such as *part-of* and *similar-to*.

The Data Element's Conceptual **Domain** can be thought of as the perception of the element's Value Domain without any physical representation. Instead of Permissible Values, only Value Meanings may be assigned. For example, the Data Element *U.S. State* might have a Value Domain that specifies postal codes or, alternatively, the full state names. In this case, we do not know how the Data Element's value will actually be represented, but we do know that its semantic interpretation must map to one of the 50 states.

Simple Data Elements can be aggregated in various ways to form more complex elements. Table 3.2.1 lists the five different ways in which primitive data elements can be combined or transformed, leading to five different derivation types.

Derivation Type	Derived Data Element	Sub Data Element	Description
Compound	Mailing Address	Street Address City State Zipcode	Grouping of Data Elements with a Display Order
Concatenation	Telephone Number	Phone Area Code Phone Exchange Phone Instrument	Grouping of Data Elements with a Display Order and Concatenation Character
Object Class	Person	Person ID Person First Name Person Last Name Person Age Person Sex	Grouping of Data Elements with optional Methods
Calculated	Person Annual Salary	Person Weekly Salary	Data Elements with a Derivation Rule (e.g., PAS = PWS * 52)
Recoded	Employment Indicator	Person Age Worked Last Week Seeking Employment	Data Elements with a Complex Derivation Rule (e.g., EI=Yes when Age >=15 and Worked Last Week = Yes)

Table 3.2.1. Derived Data Elements **(also called** Complex Data Elements**).**

In addition to the Data Elements and their related components depicted in Figure 3.2.1, the registry defines certain organizing constructs that impose semantic structure on the registry. As mentioned, a Data Element's concept can be used for grouping like elements. But this conceptual clustering is limited by the fact that each element maps to a single unambiguous concept. In practice, different categorizations may become applicable depending on the particular usage scenarios.

Part 2 of the ISO/IEC 11179 specification defines the properties that a **Classification Scheme** (CS) must exhibit. A Classification Scheme must have a **Classification Scheme Type;** examples of Classification Scheme Types are keywords, thesaurus terms, taxa and ontological terms. The Classification Scheme itself is composed of **Classification Scheme Items** (CSI), which may or may not be hierarchical. The CSI may be associated with zero or more Data Elements.

A second important organizing construct in the caDSR is a **Context**, which serves many purposes. The ISO/IEC 11179 standard defines a Context as a "designation or description of the application environment or discipline in which a name is applied or from which it originates." A Context might be an organization or business area, a clinical trial, a project, or whatever the CaDSR users decide. The idea is that all of the entities occurring in the registry are defined and managed within one or more Contexts.

Figure 3.2.2 captures the hierarchical relationships among the metadata components defined in the ISO/IEC 11179 model and in the CaDSR . The Administered Components and Administered Component Statuses appearing near the top of the diagram are described below.

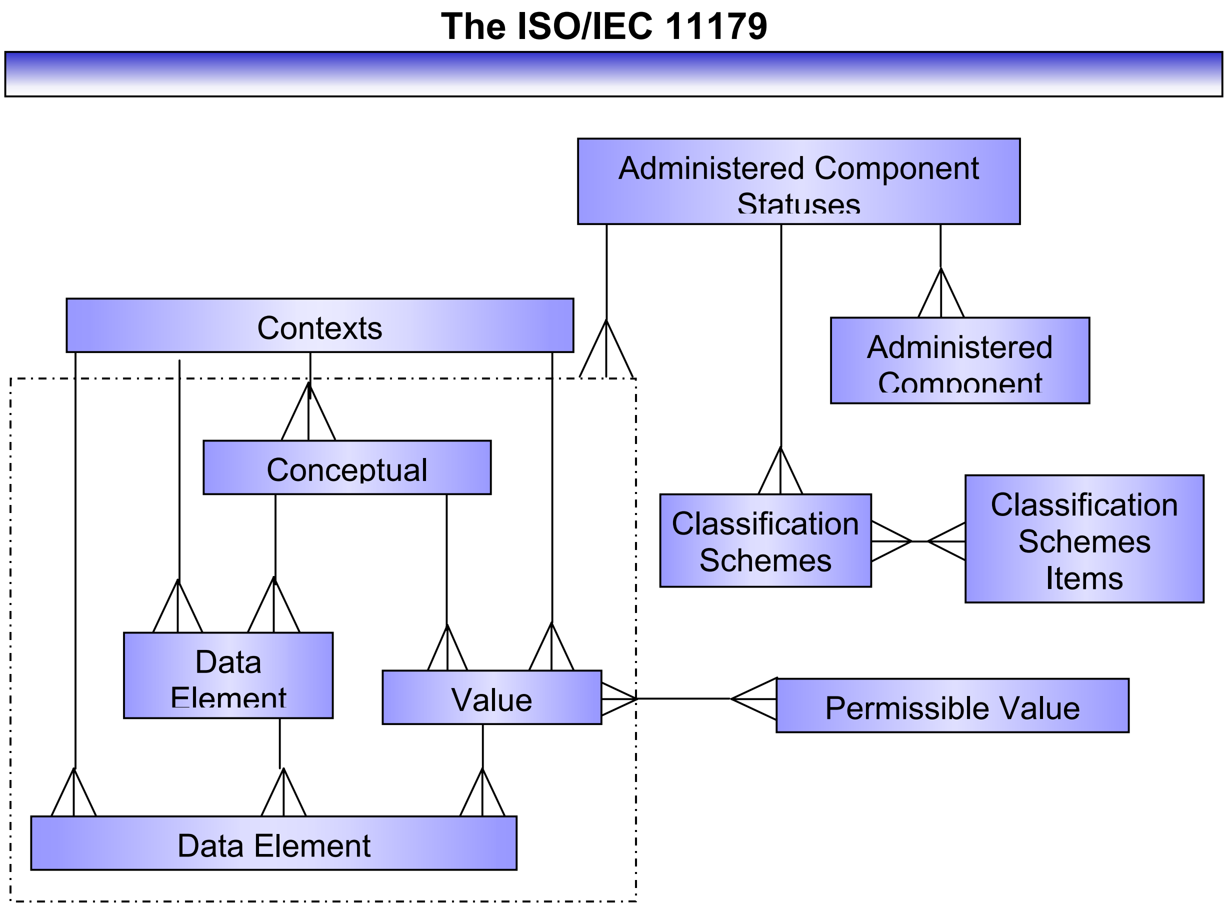


Figure 3.2.2. Hierarchical relationships among metadata components.

3.2.2 Administration and Stewardship in the CaDSR

An **Administered** Component **(AC)** is any object in the registry that requires naming, identification, and administration. The CaDSR defines all of the following as Administered Components:

- Data Elements
- Data Element Concepts
- Value Domains

- Conceptual Domains
- Classification Schemes

An Administered Component must have at least one Designation (name), and at least one Definition. While an AC may have several of these, the CaDSR requires that each AC has exactly one preferred name, preferred definition, and preferred context – all of which are maintained as direct attributes of the AC. An Administered Component in the CaDSR also has an Administration Status, specifying that the object is Legacy, Draft, Working, Reviewed, Approved, etc.

Each Data Element has a Steward who is responsible for the metadata quality of an object. This person may not have created the metadata, nor even be in charge of its maintenance, but serves as the point of contact for the Data Element. The Steward belongs to an Organization, which can be identified at any level such as Agency, Program Area, Staff Area, or Project. The CaDSR , however, does not store the hierarchical Organization chart.

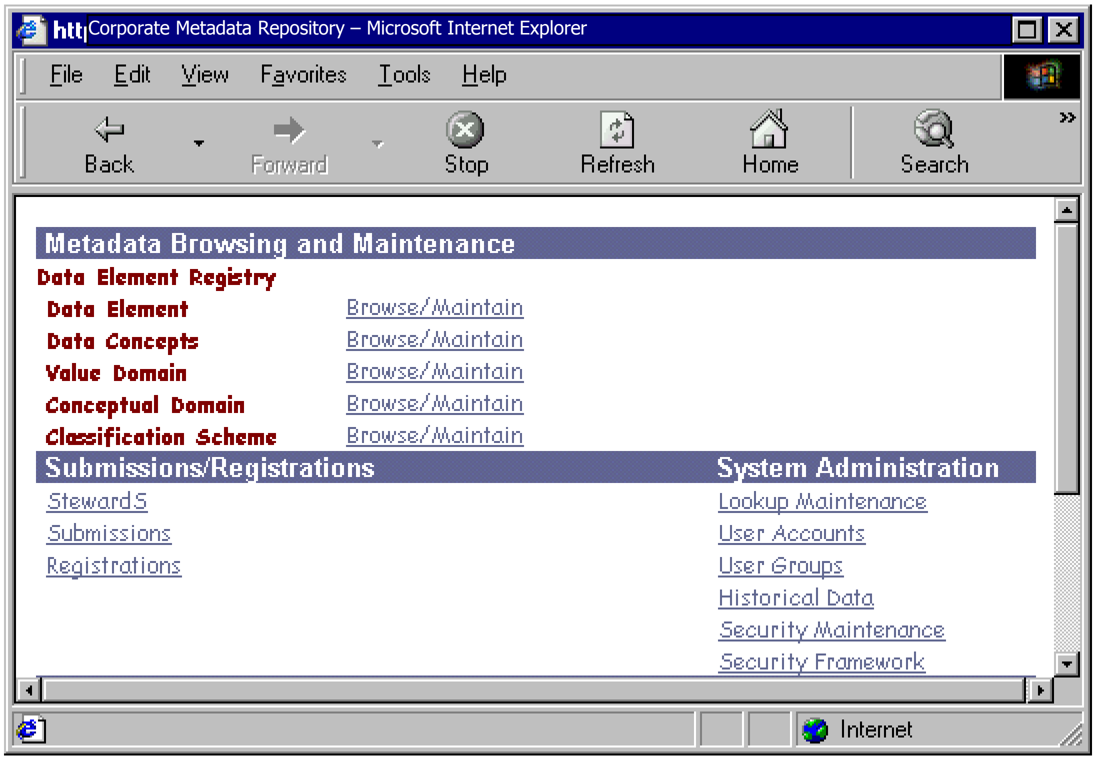
The ISO/IEC 11179 standard specifies a model for "registering" Data Elements with a Registration Authority. It is via the Registration Authority that Data Elements can be "standardized." The 11179 standard presumes registration of a given Data Element through a single authority. However, the CaDSR widens the use of the Registration meta-model, and allows Data Elements to be registered by many different Registration Authorities. The motivation for this is to permit a "staged" registration process that allows authorities to "pre-register" Data Elements locally prior to submitting them to a higher Registration Authority where the terms become part of an international standard.

3.3 The caDSR Web Interface

The Cancer Data Standards Repository (caDSR) provides a web interface for browsing, maintaining and editing the administered components stored in the central Data Element Registry. The home page for this web site (Figure 3.3.1) consists of two panels:

- **Metadata Browsing and Maintenance** provides an interface for maintaining data elements, data concepts, value domains, conceptual domains, and classification schemes;
- **Submissions/Registrations and System Administration** is provided for tasks such as the creation of new user accounts, user groups, contexts, and workflow transitions;

While the caDSR has functionality for the registration process, this capability is not currently used, and is included in this discussion for completeness only. The remainder of this section focuses instead on the **Metadata Browsing and Maintenance** capabilities.


htt Corporate Metadata Repository – Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Internet

Metadata Browsing and Maintenance

Data Element Registry

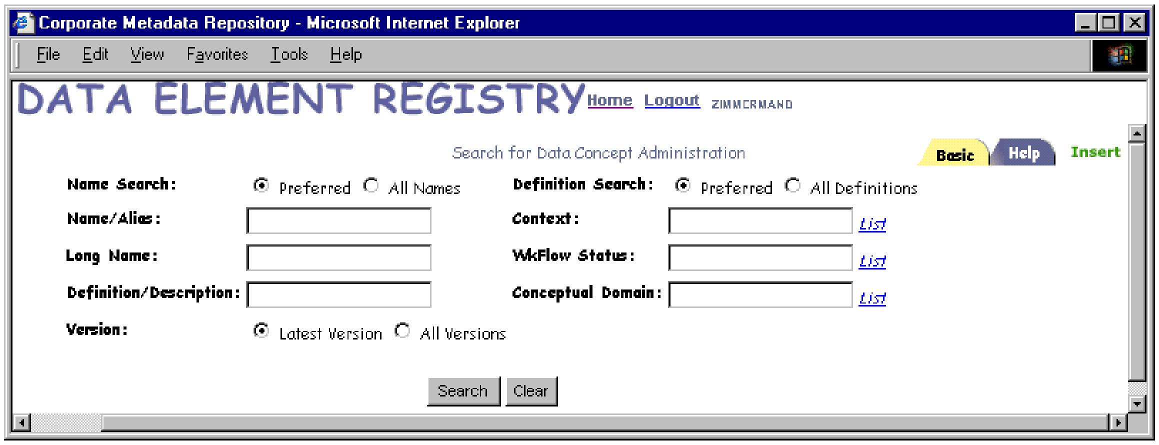
Data Element	Browse/Maintain
Data Concepts	Browse/Maintain
Value Domain	Browse/Maintain
Conceptual Domain	Browse/Maintain
Classification Scheme	Browse/Maintain

Submissions/Registrations	System Administration
StewardS	Lookup Maintenance
Submissions	User Accounts
Registrations	User Groups
	Historical Data
	Security Maintenance
	Security Framework

Figure 3.3.1. The caDSR Home page

3.3.1 The caDSR Search Interfaces

Clicking on any of the *Browse/Maintain* buttons in the top panel brings up a screen providing search and editing tools for that administered component type. The screen shot in Figure 3.3.2 shows the display that appears after clicking the *Browse/Maintain* option for data concepts. Like all of the administered components, the search criteria for a data concept include the component's name(s), definition(s), context, workflow status, and version.


Corporate Metadata Repository - Microsoft Internet Explorer

DATA ELEMENT REGISTRY Home Logout ZIMMERMAND

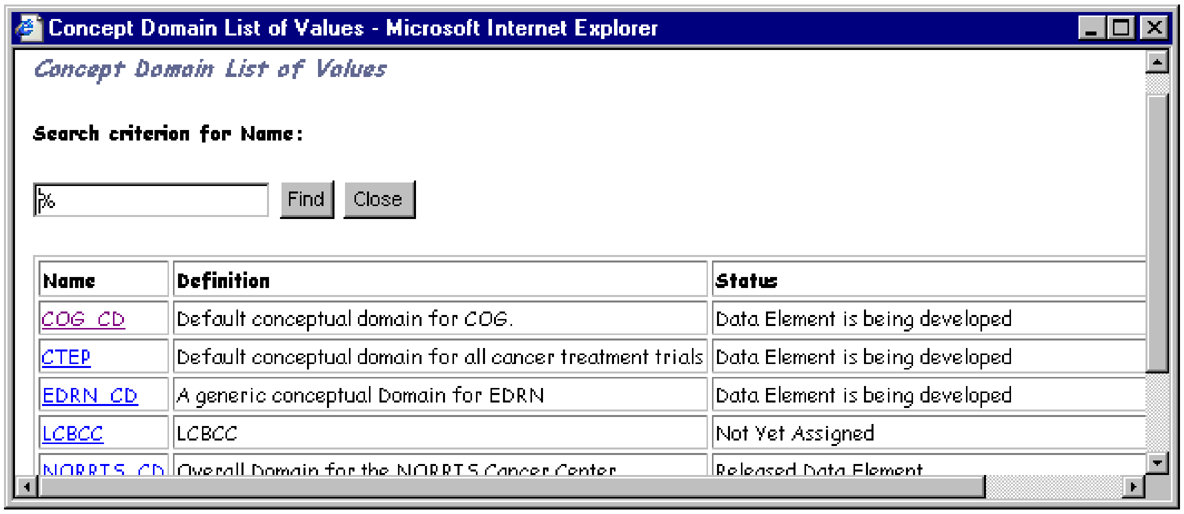
Search for Data Concept Administration

Name Search:	[✓ Preferred] All Names	Definition Search:	[✓ Preferred] All Definitions
Name/Alias:		Context:	List
Long Name:		WkFlow Status:	List
Definition/Description:		Conceptual Domain:	List
Version:	[✓ Latest Version] All Versions		

Figure 3.3.2. The Browse/Maintain screen associated with Data Concepts.

By default, preferred names and preferred definitions are used to identify matching components, but this can be overridden using the radio buttons at the top of the display. Similarly, the default of searching only the latest version can be reset to search all versions of the data registry, using the radio button on the bottom.

The only search criteria that is unique to data concepts in Figure 3.3.2 is the *Conceptual Domain* field. In general, each context defined in the caDSR has a corresponding default conceptual domain. Clicking on the *List* option alongside the text box for *Conceptual domain* will bring up a popup window listing the defined concept domains to which the user has access, as in Figure 3.3.3. Selecting any of these concept domains will in turn, fill the corresponding text box on the *Browse/Maintain* screen with that name and close the popup display. The *List* options associated with *Context* and *Workflow Status* in Figure 3.3.2 behave similarly.


Concept Domain List of Values - Microsoft Internet Explorer

Concept Domain List of Values

Search criterion for Name:

%

Name	Definition	Status
COG_CD	Default conceptual domain for COG.	Data Element is being developed
CTEP	Default conceptual domain for all cancer treatment trials	Data Element is being developed
EDRN_CD	A generic conceptual Domain for EDRN	Data Element is being developed
LCBCC	LCBCC	Not Yet Assigned
NORRTS_CD	Overall Domain for the NORRTS Cancer Center	Released Data Element

Figure 3.3.3. A list of Concept Domains to use as search criteria.

Each of the other administered components, in addition to the standard search criteria shown in Figure 3.3.2, has its own supplemental criteria which are unique to that component type. Data elements have *Data Concept* and *Value Domain* fields; concept domains have a *Value Meaning* field; and classification schemes have a *Classification*

S*cheme Type* field. Value domains have the largest number of associated search criteria fields, which include: *Data Type, Format, Domain Type*, *Unit of Measure,* and *Character Set.* These additional search criteria fields have popup windows associated with them to provide the user with the list of possible values.

3.3.1.1 Basic Search

All of the *Browse/Maintain* screens support basic search, with interfaces similar to that shown in Figure 3.2.2. Basic search allows the user to search for components by name, definition, context, workflow status, and/or version.

The first name given to an Administered Component is stored as its preferred name. When searching by name, candidate matches are ranked differently depending on whether the user has selected the "Preferred Name" or "All Names" radio button. If the "Preferred Name" button is selected, then the name entered in the *Name/Alias* field will only be searched against preferred names; otherwise, all names will be searched.

It is also possible to search for a data element using its *CDE Identifier*. The CDE Identifier is a unique seven-digit number assigned to each data element, with the name of the disease associated with that element appended to the number (e.g. "2001101LUNG"). Search by CDE Identifier can be done by clicking on the "All Names" radio button and entering the partial or full CDE Identifier into the "Name/Alias" field. Wildcard matching can also be used. For example, "2001101%" can be used to retrieve "2001101LUNG."

As with preferred names, the first definition given to an Administered Component is also recorded as the *preferred* definition, and selecting the "Preferred Definition" radio button will limit the matching to preferred definitions only. The remaining search fields common to all of the basic search screens are:

- *Long Name*: This is the Long Name given to an Administered Component. In some implementations, this field may be used for a name with an agreed upon naming convention (such as recommended by ISO/IEC 11179).
- *Workflow Status***:** This is the administrative (workflow) status of an Administered Component (such as Draft, Reviewed, Approved, Released). Select the *List* icon to get the list of values.
- *Context***:** This is the context of the Administered Component's preferred name (e.g. the first Context associated with an AC).
- *Latest Version or All Versions***:** The caDSR allows you to retrieve just the latest version of an AC or all versions of the AC. An Administered Component is uniquely identified by its preferred Name, preferred Context, and Version.

3.3.1.2 Full Text Search

The interfaces for classification schemes and data concepts support only basic search. The other components, i.e., data elements, value domains, and concept domains, have search interfaces which also support *full-text* search. Full Text searching (Figure 3.3.4) allows you to enter unstructured, full text information about a given administered component. This full text combines the name, long name, description fields, and other relevant fields. If there is a hit on the search term in any of the combined text, the administered component is returned.

Full text searches are case insensitive, and support wildcard (%) matching. Specific search options are provided via the drop-down selection boxes surrounding the textboxes, and up to three text strings can be specified. These strings can be combined differently according to the selected Boolean operator.

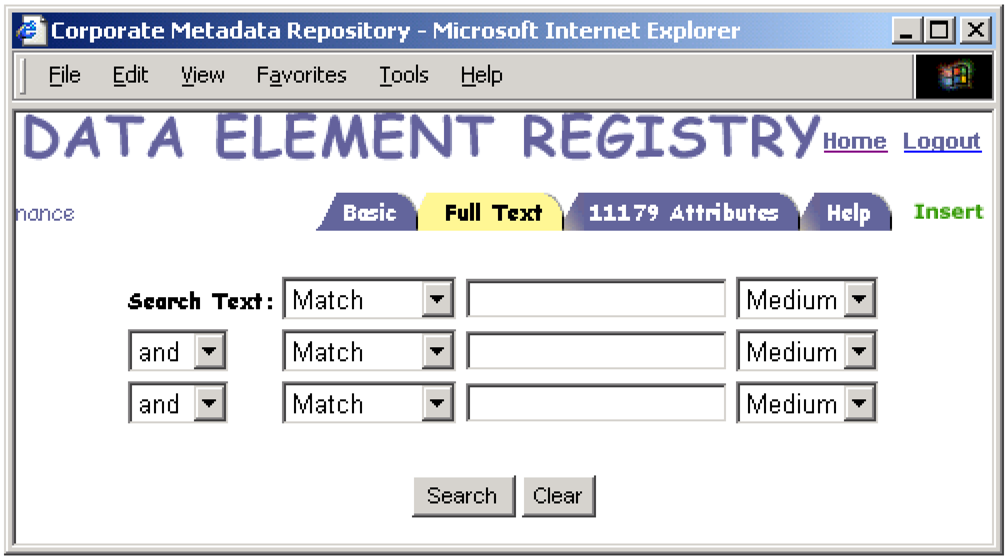


Figure 3.3.4. Full Text Search Screen

The leftmost drop-down selection boxes specify the Boolean operators to be used in combining the expressions occurring in the preceding and subsequent textboxes. The choices include *and, or, not,* and *near,* where:

- *and* evaluates to TRUE if the candidate component matches both expressions;
- *or* evaluates to TRUE if the candidate component matches either expression;
- *not* evaluates to TRUE if the candidate component matches the first expression but does not match the second expression;
- *near* evaluates to TRUE if the first expression occurs "near" the second expression in the candidate component's matching terms.

The Boolean operators are evaluated in top-to-bottom order, and there is no precedence to the operators other than that defined by position.

The next drop-down selection box ("Match" in Figure 3.3.4) allows the user to specify further constraints on how the matching is done; options include *Match, Starts with, Stem,* and *Soundex.* The default, *Match,* specifies that unconstrained lexicographic matching should be used. The other options are as follows:

- *Starts with* searches for text that starts with the expression provided in the textbox;
- *Stem* searches for text that shares the same stem as the textbox expression;

- *Soundex* searches for text that “approximately” sounds like the search expression (e.g. “reed” and “read”, “wage” and “age”, etc.). Soundex matching is based on a very simple encoding which approximately captures the significant phonemes in a string.

Finally, each search condition can also be prioritized as *High, Medium* (the default), or *Low*. Hits on high priority expressions will be returned first, followed by medium, and then low priority expressions. Only those components whose text satisfy all of the search criteria (as combined by the selected Boolean operators) are returned by the full text search.

3.3.1.3 11179 Attribute Search

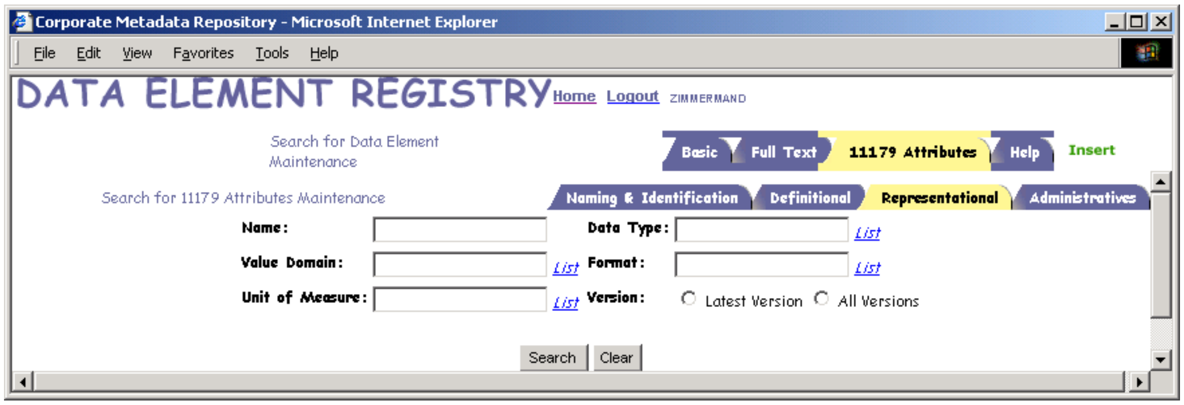


Figure 3.3.5. 11179 Attributes Search.

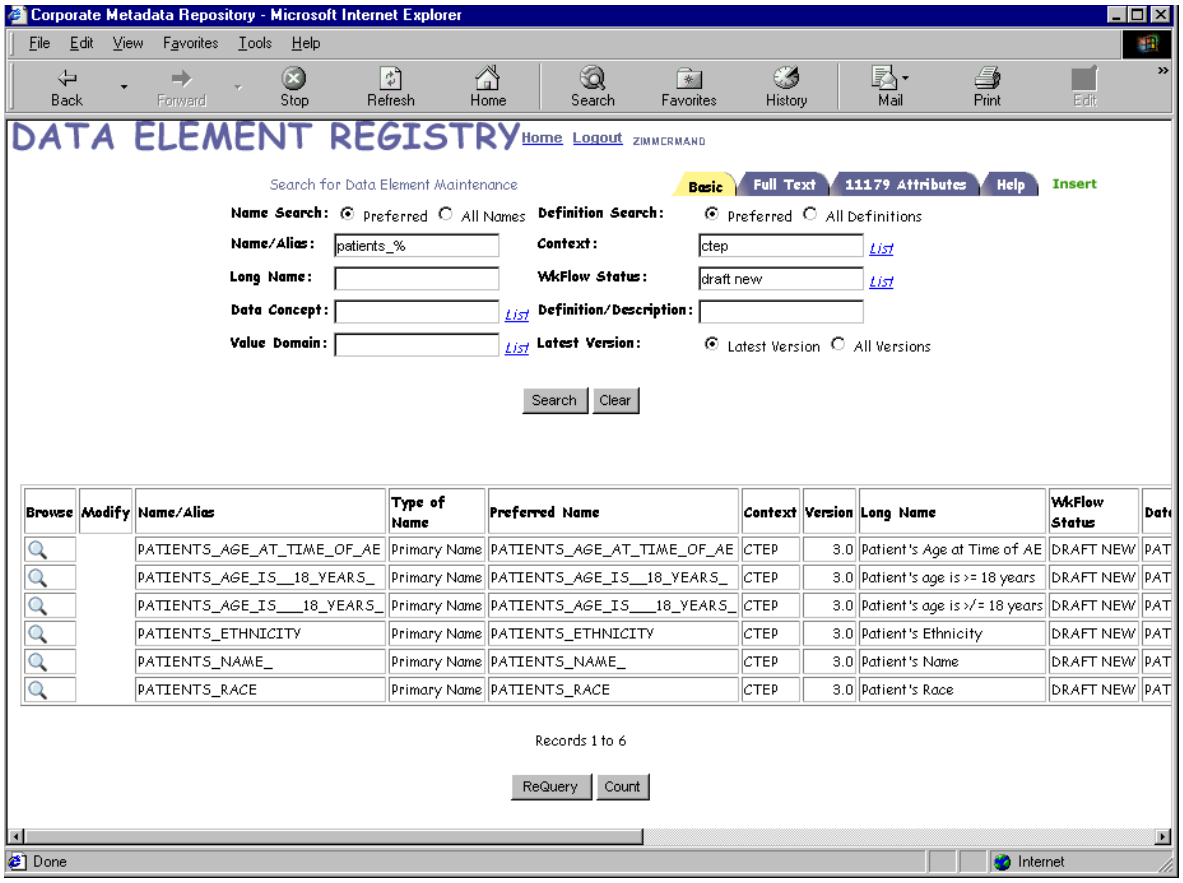
The search interface for data elements provides one additional mode of search – the *11179 Attribute Search*. The 11179 Attributes search interface (Figure 3.5.5) allows the user to search by the ISO/IEC 11179 attributes in four categories: *Naming and Identification, Definitional, Representational,* and *Administrative*. The first two categories simply provide subsets of the search fields available with the *Basic Search* interface. Figure 3.5.5 shows the interface when the third tab, *Representational*, has been selected. As shown there, this interface allows the search to drill down into the data element’s value domain attributes without explicitly using the value domain’s search interface.

3.3.1.4 Search Results

Search results are returned according to the specified search criteria and the individual user’s privileges, with the results listed in a table (Figure 3.3.6) immediately below the search form. Like all search result tables, the leftmost column of the table in Figure 3.3.6 displays a *Browse* icon in the form of a magnifying glass. Clicking on that icon brings up the browse page (Figure 3.3.7) for the administered component in that row of the table.

If the user has editing permissions for that record, then the second column displays a *Modify* icon in the form of a pencil. Alternatively, if the user does not have editing privileges, the second column is empty, as in Figure 3.3.6. Note that if the user has neither browse nor edit permissions on a given component, that record will not be displayed in the results table – even if its attributes otherwise matched the search criteria.

The remaining columns in the results table correspond to the fields in the component type’s search form. Thus, for a data element, the fields are: *Name/Alias, Type of Name, Preferred Name, Context, Version, Long Name, Workflow Status, Data Concept,* *Definition, Type of Definition,* and *Value Domain*. Similarly the results table for a value domain will contain all of these fields except *Data Concept* and *Value Domain,* and will add the fields: *Domain Type, Data Type, Unit of Measure, Format,* and *Character Set*.


Corporate Metadata Repository - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
DATA ELEMENT REGISTRY
Home Logout ZIMMERMAND
Basic Full Text 11179 Attributes Help Insert
Search Clear
ReQuery Count
Done
Internet

Search for Data Element Maintenance

Name Search:	(● Preferred) ○ All Names	Definition Search:	(● Preferred) ○ All Definitions
Name/Alias:	patients_%	Context:	ctep List
Long Name:		WkFlow Status:	draft new List
Data Concept:	List	Definition/Description:	
Value Domain:	List	Latest Version:	(● Latest Version) ○ All Versions

Browse	Modify	Name/Alias	Type of Name	Preferred Name	Context	Version	Long Name	WkFlow Status	Dat
🔍		PATIENTS_AGE_AT_TIME_OF_AE	Primary Name	PATIENTS_AGE_AT_TIME_OF_AE	CTEP	3.0	Patient's Age at Time of AE	DRAFT NEW	PAT
🔍		PATIENTS_AGE_IS__18_YEARS_	Primary Name	PATIENTS_AGE_IS__18_YEARS_	CTEP	3.0	Patient's age is >= 18 years	DRAFT NEW	PAT
🔍		PATIENTS_AGE_IS___18_YEARS_	Primary Name	PATIENTS_AGE_IS___18_YEARS_	CTEP	3.0	Patient's age is >/= 18 years	DRAFT NEW	PAT
🔍		PATIENTS_ETHNICITY	Primary Name	PATIENTS_ETHNICITY	CTEP	3.0	Patient's Ethnicity	DRAFT NEW	PAT
🔍		PATIENTS_NAME_	Primary Name	PATIENTS_NAME_	CTEP	3.0	Patient's Name	DRAFT NEW	PAT
🔍		PATIENTS_RACE	Primary Name	PATIENTS_RACE	CTEP	3.0	Patient's Race	DRAFT NEW	PAT

Records 1 to 6

Figure 3.3.6. The Search Results Table

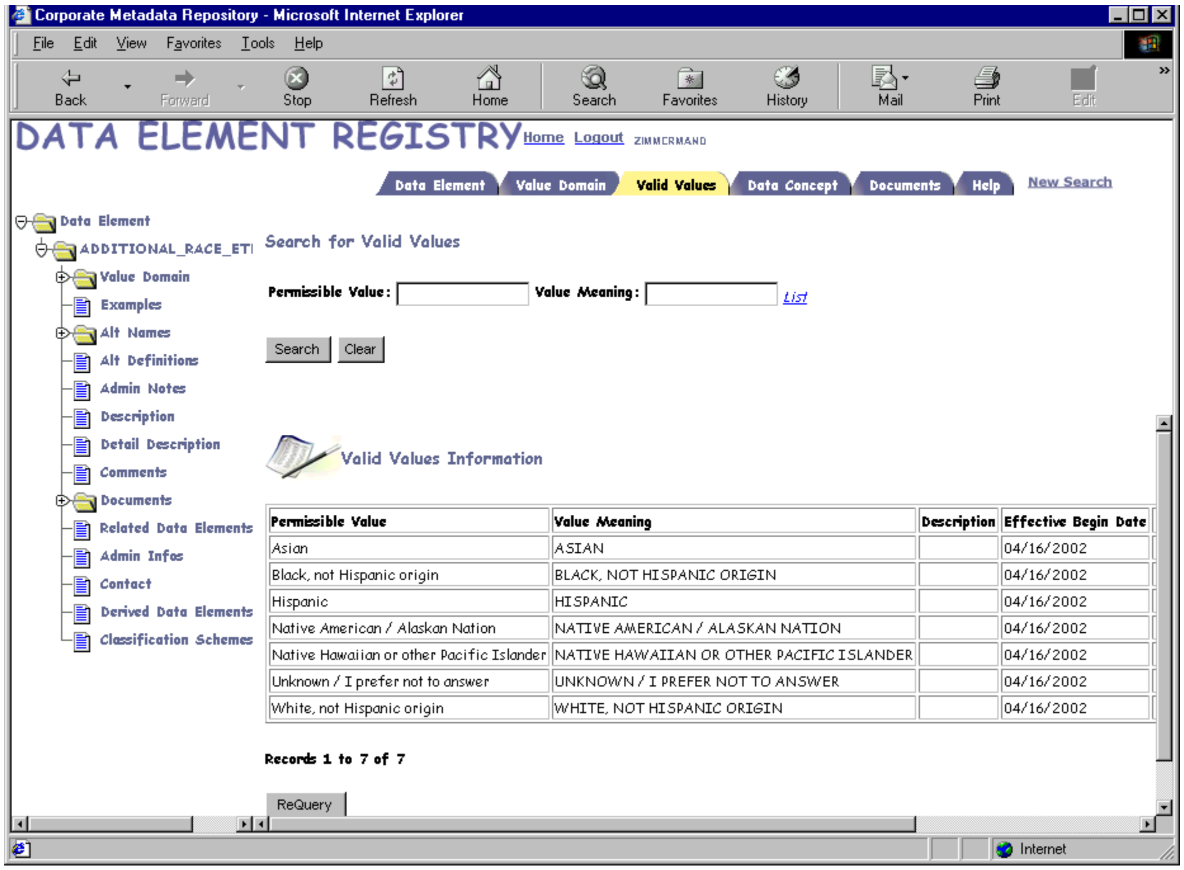
Large search result tables also have navigation buttons at the bottom of the list. If the result set is large, you may need to scroll through sets of records using the *Next, Previous, First*, and *Last* buttons. The *ReQuery* button re-executes the same query; the *Count* button shows the total number of records that met the search criteria. A user may execute a new search by pressing *Clear*, entering new search criteria, and pressing *Search* again.

Figure 3.3.7 shows the browse screen for the ADDITIONAL_RACE_ETHNICITY data element. The *Valid Values* tab has been selected, and the main frame shows the table of permissible values for an element of this type. The screen also provides access to information about the *Value Domain, Data Concept*, and any documents associated with this element. Each node in the tree-structured display on the left is selectable, and provides an alternate way of navigating through the information associated with the data element.

3.3.1.5 Summary of Search Screen Behaviors

- The Search criteria fields are not case sensitive;
- If search criteria are provided in multiple fields, a logical AND is applied – i.e. the matched component must satisfy all of the criteria;

- When no criteria are specified, the search retrieves all records for that component type which the user has access to;
- The percent sign (%) may be used for wildcard matching;
- *List* icons allow the user to choose from a list of valid values for the given field.
- The *Search* button invokes the actual search;
- The *Clear* button resets all fields in the search form;



Permissible Value	Value Meaning	Description	Effective Begin Date
Asian	ASIAN		04/16/2002
Black, not Hispanic origin	BLACK, NOT HISPANIC ORIGIN		04/16/2002
Hispanic	HISPANIC		04/16/2002
Native American / Alaskan Nation	NATIVE AMERICAN / ALASKAN NATION		04/16/2002
Native Hawaiian or other Pacific Islander	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER		04/16/2002
Unknown / I prefer not to answer	UNKNOWN / I PREFER NOT TO ANSWER		04/16/2002
White, not Hispanic origin	WHITE, NOT HISPANIC ORIGIN		04/16/2002

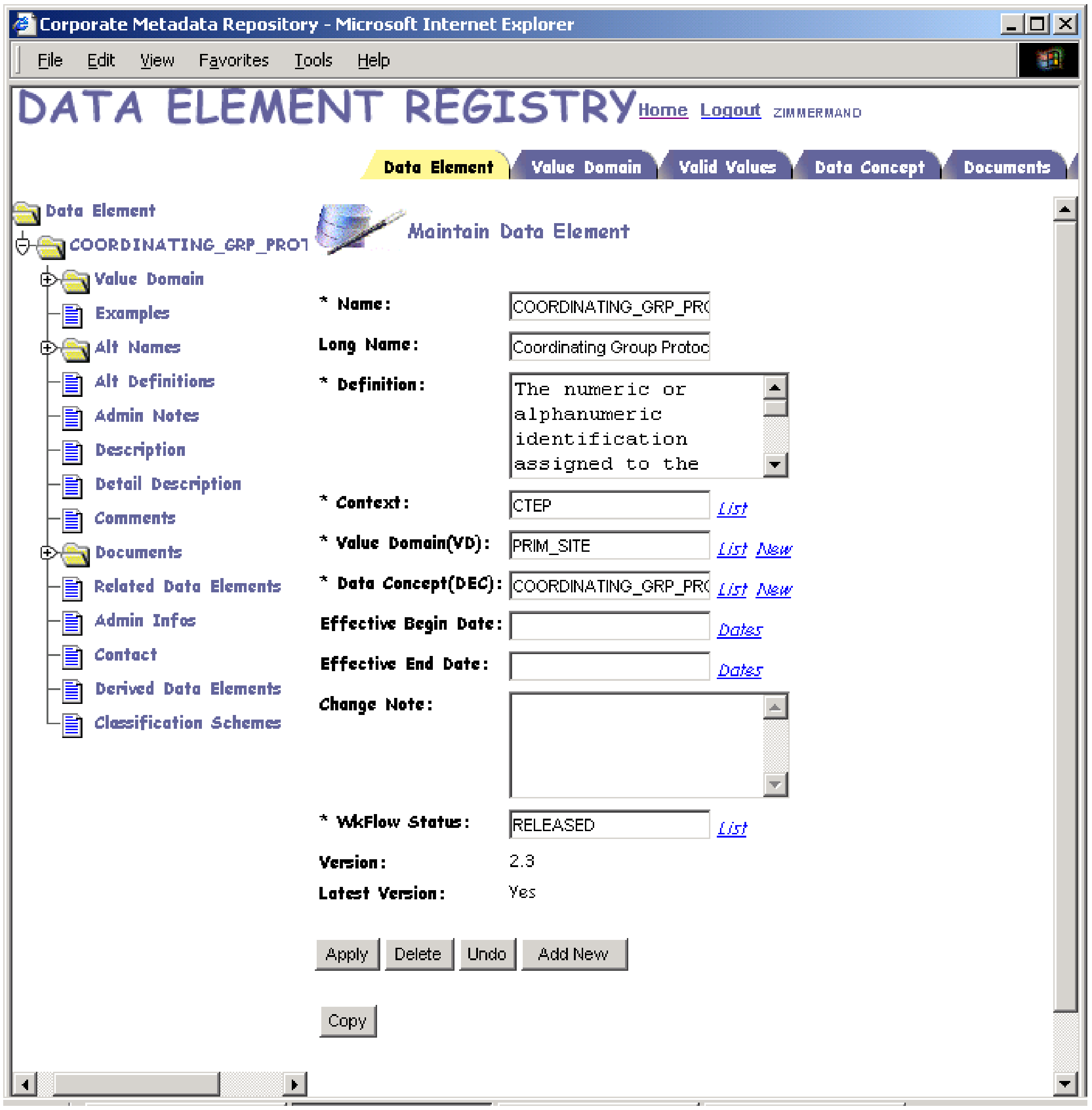
Records 1 to 7 of 7

Figure 3.3.7. Browsing Screen for ADDITIONAL_RACE_ETHNICITY

3.3.2 Maintenance Screens for Administered Components

Maintenance screens combine the appearance of browsing screens with the functionality of the displays used to create new components. Figure 3.3.8 shows the maintenance screen for a data element named COORDINATING_GRP_PROTOCOL_NUM.

The maintenance screen is reached by clicking the pencil icon displayed in the second column of the results table. Editing capabilities include adding, updating, and deleting the information fields shown in the maintenance screen. The tree icons in the left panel provide access to editing screens for the corresponding attributes of the component. The maintenance screen for a particular administered component is only accessible to users having *Update* and *Delete* privileges.


Corporate Metadata Repository - Microsoft Internet Explorer

File Edit View Favorites Tools Help

DATA ELEMENT REGISTRY Home Logout ZIMMERMAND

Data Element | Value Domain | Valid Values | Data Concept | Documents

Data Element
- COORDINATING_GRP_PROT
  - Value Domain
  - Examples
  - Alt Names
  - Alt Definitions
  - Admin Notes
  - Description
  - Detail Description
  - Comments
  - Documents
  - Related Data Elements
  - Admin Infos
  - Contact
  - Derived Data Elements
  - Classification Schemes

Maintain Data Element

* Name:	COORDINATING_GRP_PR(	
Long Name:	Coordinating Group Protoc	
* Definition:	The numeric or alphanumeric identification assigned to the	
* Context:	CTEP	List
* Value Domain(VD):	PRIM_SITE	List New
* Data Concept(DEC):	COORDINATING_GRP_PR(	List New
Effective Begin Date:		Dates
Effective End Date:		Dates
Change Note:		
* WkFlow Status:	RELEASED	List
Version:	2.3	
Latest Version:	Yes	

Apply Delete Undo Add New

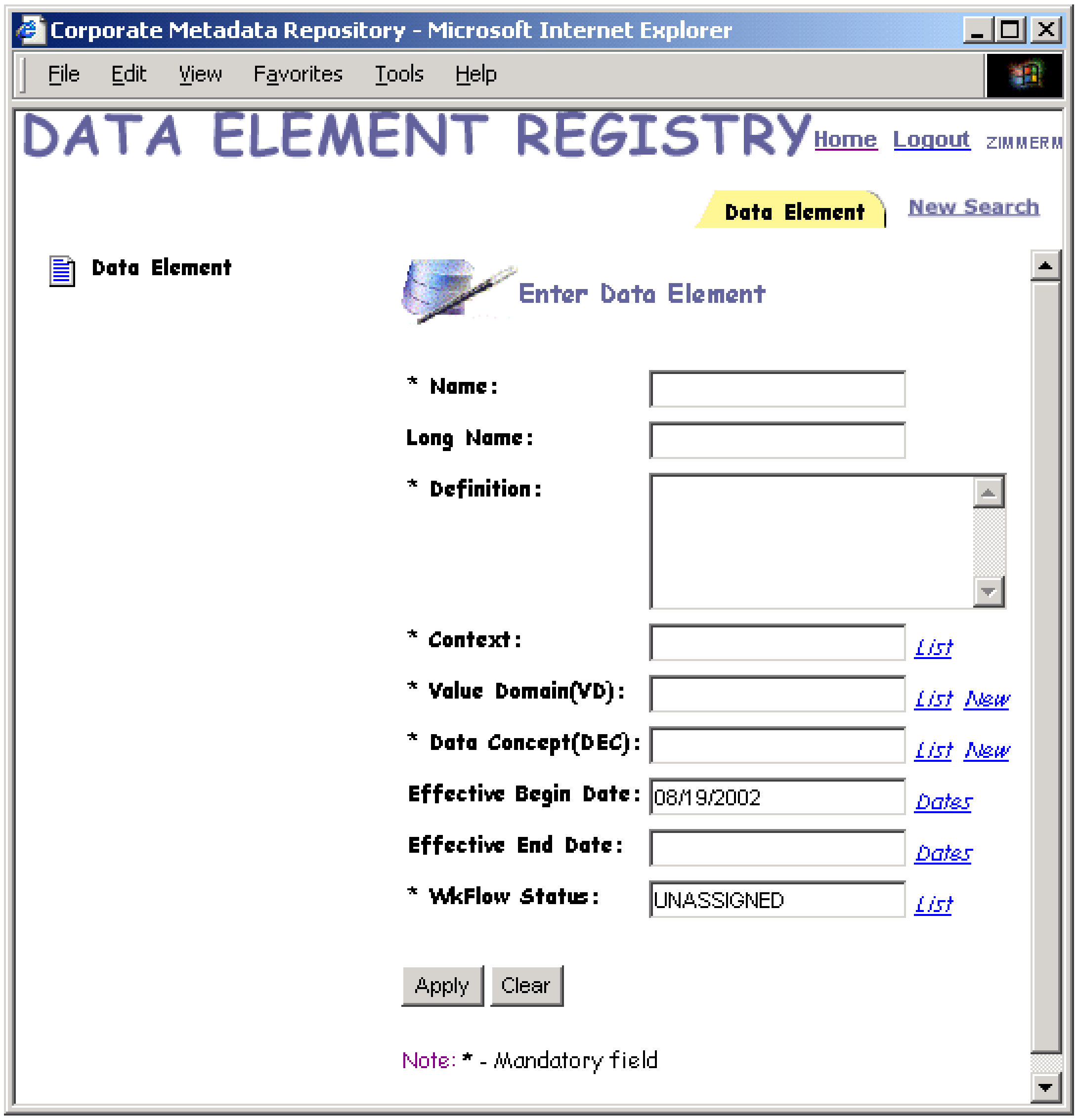
Copy

Figure 3.3.8. Maintenance screen.

3.3.3 Creating Administered Components

Each of the search screens described in Section 3.3.1 include a rightnost tab with the keyword *insert* displayed on it. Clicking on this tab for the data element search interface brings up the screen shown in Figure 3.3.9 below. Alternatively, from the maintenance screen for an adinistered component (e.g. Figure 3.3.8), you can press the *Add New* button (bottom, right) to create a new component.

Figure 3.3.9 shows the interface for creating a new data element component. As some of a new component's attribute values may themselves be administered components (e.g. the Value Domain or Data Concept for a Data Element), the interface provides mechanisms for selecting from the set of currently defined components, as well as recrusively creating a new component on-the-fly. The *New* icon allows you to (1) create the new component, and (2) subsequently assign it via the *List* icon.


Corporate Metadata Repository - Microsoft Internet Explorer
File Edit View Favorites Tools Help
DATA ELEMENT REGISTRY
Home Logout ZIMMERM
Data Element
New Search
Data Element
Enter Data Element

* Name:		
Long Name:		
* Definition:		
* Context:		List
* Value Domain(VD):		List New
* Data Concept(DEC):		List New
Effective Begin Date:	08/19/2002	Dates
Effective End Date:		Dates
* WkFlow Status:	UNASSIGNED	List

Note: * - Mandatory field

Figure 3.3.9. Creating a new data element component

The caDSR interface requires that most of the mandatory attributes of ISO/IEC 11179 must be supplied. Exceptions are the submission, registration, and stewardship assignments, which are not included in the form. Required fields are indicated by an asterisk preceding the attribute name. If the value for a mandatory field is not currently known, enter 'UNASSIGNED'. The ISO/IEC 11179 Compliancy Test will look for and warn users of mandatory fields with an entry of 'UNASSIGNED'.

Once the administered component has been created, the application will display the maintenance screen, where you can add additional relationships and information for that component.

3.4 Overview of the caDSR Data Model

The last section of this chapter contains a complete diagram of all of the tables in the cancer Data Standards Repository (caDSR). This section serves as an introduction to the full-scale data model, as it presents reduced views of the system

Figure 3.2.2 of Section 3.2 illustrated the hierarchical relationships among the ISO/IEC 11179 metadata components defined in the CaDSR data model. The tables representing these components are listed in Table 3.4.1, and their relationships to one another are summarized in Figure 3.4.1.

Component Name	Table Name
Data Elements	DATA_ELEMENTS (DE)
Value Domains	VALUE_DOMAINS (VD)
Data Element Concepts	DATA_ELEMENT_CONCEPTS (DEC)
Conceptual Domains	CONCEPTUAL_DOMAINS (CD)
Classification Schemes	CLASSIFICATION_SCHEMES (CS)
Administered Components	ADMINISTERED_COMPONENTS (AC)
Contexts	CONTEXTS (CONTE)
Classification Schemes Items	CLASS_SCHEME_ITEMS (CSI)
Administered Component Statuses	AC_STATUS_LOV (ASL)

Table 3.4.1. The metadata component tables in the data model

At the top level, the CONTEXTS and AC_STATUS_LOV tables are linked to all of the remaining tables – with the exception of the CLASS_SCHEME_ITEMS table. These links correspond to foreign keys defined in those tables. For example, the VALUE_DOMAINS table has a foreign key named *asl_name*, which corresponds to the primary key (*asl_name*) in the AC_STATUS_LOV table. The link connecting the tables is labeled by catenating the abbreviated names of the two tables and appending the suffix _FK to the result.

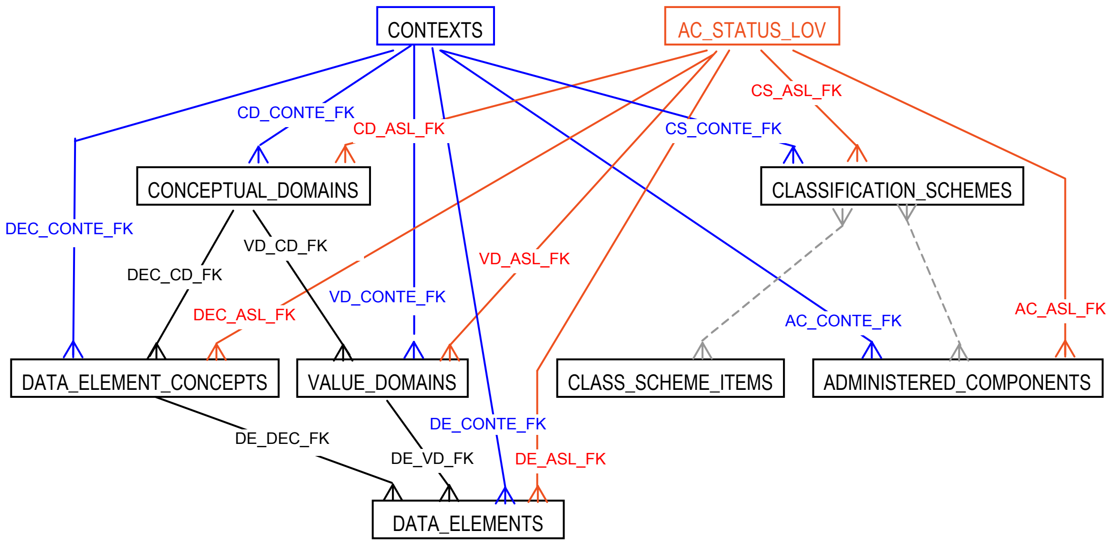


Figure 3.4.1. An abstract view of the caDSR data model

Note that these links are unidirectional; they define many-to-one relations – not many-to-many relations. Thus for example, each data element maps to a single context, but many can map to the same single context.

The links between the CLASSIFICATION_SCHEMES, CLASS_SCHEME_ITEMS, and ADMINISTERED_COMPONENTS tables are an abstraction of the actual data model. Here, many-to-many relations are defined between these tables, but not directly. As illustrated in Figure 3.4.2, two intermediate tables, CS_CSI and AC_CSI are defined which implement these relations while maintaining normalized tables.

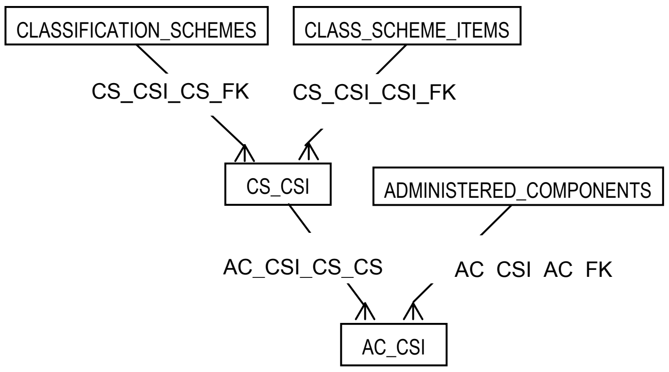


Figure 3.4.2. Auxillary tables used to implement many-to-many relations

The data model uses simple naming conventions to facilitate the interpretation of table and key names:

- Table names are abbreviated as the catenation of the first letter of each word when they are incorporated into the names of auxillary tables or key names. For example, the VALUE_DOMAINS table is abbreviated as VD. Exceptions are one-word names, such as CONTEXTS. In these cases, the full name is truncated, as in CONTE_IDSEQ, the primary key occurring in the CONTEXTS table.
- The primary key for a table is in most cases generated by concatenating the abbreviated table name with the string IDSEQ. For example, the primary key for VALUE_DOMAINS is VD_IDSEQ. The alternative convention is to append the string NAME in place of IDSEQ.
- The foreign key name is identical to the name of the primary key in the foreign table it references. Thus the foreign key CONTE_IDSEQ is used in the VALUE_DOMAINS table to reference the CONTEXTS table.
- In diagrams of the data model, foreign keys are represented as many-to-one relations. The link connecting the two tables is labeled by concatenating the names of the two tables and appending the suffix _FK to the result. For example, the VALUE_DOMAINS table contains the foreign key CD_IDSEQ, which references the primary key in the. CONCEPTUAL_DOMAINS table. In Figure 3.4.1, this link is labeled VD_CD_FK.
- Table names ending in _LOV contain a list of values.
- Table names ending in _RECS store relationships between records occuring in the same (external) table.

Figure 3.4.3 elaborates on the reduced view of the data model, introducing several additional auxillary tables. In two cases several tables have been collapsed into a single "multi-table" icon in order to simplify the diagram. Specifically, the tables PROPERTIES_LOV and OBJECT_CLASSES_LOV are collapsed into one, as are the tables DATA_TYPES_LOV, FORMATS_LOV, CHARACTER_SET_LOV, and UNIT_OF_MEASURES_LOV. In these cases, the collapsed tables share the same connectivities.

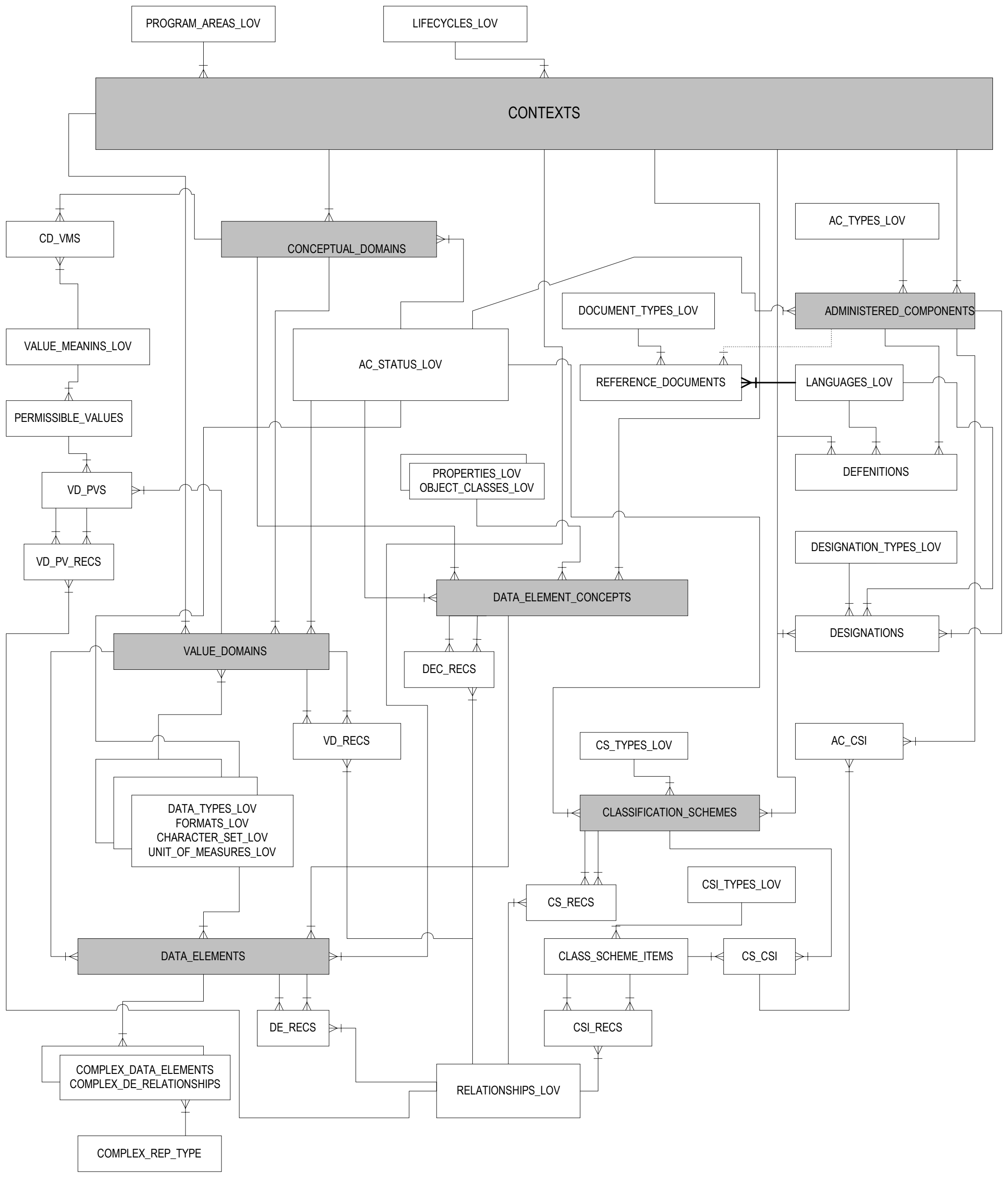


Figure 3.4.3. Auxillary tables used to implement many-to-many relations.

Several of the tables in Figure 3.4.3 store relationships between records occuring in the same table, and have names ending in _RECS. Each of these has two foreign keys into the table over which the relations are defined.

The next section provides a catalog of the caDSR tables, and the final section of this chapter contains the complete data model, including all tables along with their relations and field names. A separate document, The caCORE 1.0 caDSR Database API, provides detailed information on the PL/SQL stored procedures.

3.5 The caDSR Table Catalog

The definitions used below should be interpreted as follows:

- *Foreign keys* lists the names of the tables into which the current table holds foreign keys.
- *Referenced by* lists the names of the tables which hold foreign keys into the current table
- *Linked to* lists the tables to which the current table has indirect relations which are implemented by intermediary tables.

AC_CSI: links administered components (AC) to classification scheme items (CSI).
Foreign keys: ADMINISTERED_COMPONENTS, CS_CSI
Referenced by: none

AC_STATUS_LOV (ASL): stores the administrative status of administered components (AC).
Foreign keys: none.
Referenced by: ADMINISTERED_COMPONENTS, CLASSIFICATION_SCHEMES, CONCEPTUAL_DOMAINS, DATA_ELEMENTS, DATA_ELEMENT_CONCEPTS, VALUE_DOMAINS

AC_TYPES_LOV (ATL): stores the types of administered components and their possible values.
Foreign keys: none.
Referenced by: ADMINISTERED_COMPONENTS

ADMINISTERED_COMPONENTS (AC): stores information about objects that require naming and administration (administered components).
Foreign keys: CONTEXTS, AC_STATUS_LOV, AC_TYPES_LOV.
Referenced by: REFERENCE_DOCUMENTS, DEFINITIONS, DESIGNATIONS.
Linked to: CLASSIFICATION_SCHEMES (AC_CSI)

CD_VMS: stores the value meanings (VMS) for a conceptual domain (CD).
Foreign keys: VALUE_MEANINGS_LOV, CONCEPTUAL_DOMAINS
Referenced by: none

CHARACTER_SET_LOV (CSL): stores character set values (e.g. ASCII, Unicode) for VDs.
Foreign keys: none.
Referenced by: VALUE_DOMAINS

CLASS_SCHEME_ITEMS (CSI): Stores classification scheme items
Foreign keys: CSI_TYPES_LOV
Referenced by:CSI_RECS, CS_CSI
Linked to: CLASSIFICATION_SCHEMES (CS_CSI), ADMINISTERED_COMPONENTS (AC_CSI)

CLASSIFICATION_SCHEMES (CS): stores classification schemes used to classify DEs.
Foreign keys: CONTEXTS, AC_STATUS_LOV, CS_TYPES_LOV.
Referenced by: CS_RECS, CS_CSI.
Linked to: CLASS_SCHEME_ITEMS (CS_CSI), ADMINISTERED_COMPONENTS (AC_CSI)

COMPLEX_DATA_ELEMENTS: stores information about whether or not a data element is complex (derived from a more primitive element).
Foreign keys: COMPLEX_REP_TYPE_LOV, DATA_ELEMENTS
Referenced by: COMPLEX_DE_RELATIONSHIPS

COMPLEX_DE_RELATIONSHIPS: stores information regarding how a data element was derived.
Foreign keys: COMPLEX_DATA_ELEMENTS, DATA_ELEMENTS
Referenced by: none

COMPLEX_REP_TYPE_LOV: stores the types of derivations for data elements, e.g., 'CALCULATED', 'COMPLEX RECODE', 'COMPOUND', 'CONCATENATION', etc..
Foreign keys: none.
Referenced by: COMPLEX_DATA_ELEMENTS

CONCEPTUAL_DOMAINS (CD): stores the conceptual domain components which define the semantic interpretations of data elements.
Foreign keys: CONTEXTS
Referenced by: CD_VMS, DATA_ELEMENT_CONCEPTS, VALUE_DOMAINS.

CONTEXTS (CONTE): stores information about contexts, components describing an application environment or a discipline in which a name is applied.
Foreign keys: LIFECYCLES_LOV, PROGRAM_AREAS_LOV.
Referenced by: ADMINISTERED_COMPONENTS, CLASSIFICATION_SCHEMES, CONCEPT_DOMAINS, DATA_ELEMENTS, DATA_ELEMENT_CONCEPTS, VALUE_DOMAINS, VD_PVS, DESIGNATIONS, DEFINITIONS.

CS_CSI: stores the relationships between classification schemes (CS) and classification scheme items (CSI).
Foreign keys: CLASSIFICATION_SCHEMES, CLASS_SCHEME_ITEMS
Referenced by: AC_CSI

CS_RECS: defines the relationships between two classification schemes (*Part-of, Similar-To*, etc).
Foreign keys: RELATIONSHIPS_LOV, CLASSIFICATION_SCHEMES
Referenced by: none

CS_TYPES_LOV: specifies the type of a classification scheme (CS).
Foreign keys: none.
Referenced by: CLASSIFICATION_SCHEMES

CSI_RECS: defines relationships between classification scheme items (*Part-of, Similar-To*, etc).
Foreign keys: RELATIONSHIPS_LOV, CLASS_SCHEME_ITEMS

Referenced by: None

CSI_TYPES_LOV: specifies the type of a classification scheme item (CSI).
Foreign keys: none.
Referenced by: CLASS_SCHEME_ITEMS

DATA_ELEMENT_CONCEPTS (DEC):
Foreign keys: CONTEXTS, CONCEPTUAL_DOMAINS, AC_STATUS_LOV, PROPERTIES_LOV, OBJECT_CLASSES_LOV
Referenced by: DEC_RECS,DATA_ELEMENTS

DATA_ELEMENTS (DE): stores the actual data elements.
Foreign keys: AC_STATUS_LOV, CONTEXTS, DATA_ELEMENT_CONCEPTS**,** VALUE_DOMAINS.
Referenced by: DE_RECS, COMPLEX_DATA_ELEMENTS, COMPLEX_DE_RELATIONSHIPS.

DATATYPES_LOV: stores values for datatypes (e.g. character, number, date) for VDs.
Foreign keys: none.
Referenced by: VALUE_DOMAINS

DE_RECS: defines relationships (Part-of, Similar-To etc) between data elements.
Foreign keys: RELATIONSHIPS_LOV, DATA_ELEMENTS
Referenced by: none

DEC_RECS: defines relationships (Part-of, Similar-To etc) between data element concepts.
Foreign keys: RELATIONSHIPS_LOV, DATA_ELEMENT_CONCEPTS
Referenced by: none

DEFINITIONS: stores alternate definitions for administered components.
Foreign keys: LANGUAGES_LOV, CONTEXTS, ADMINISTERED_COMPONENTS
Referenced by: none

DESIGNATIONS stores alternate designations or names for administered components.
Foreign keys: LANGUAGES_LOV, DESIGNATIONS_TYPE_LOV, CONTEXTS, ADMINISTERED_COMPONENTS
Referenced by: none

DESIGNATIONS_TYPES_LOV: stores the types of possible designations, such as synonym.
Foreign keys: none.
Referenced by: DESIGNATIONS

DOCUMENT_TYPES_LOV: stores the possible types of reference documents.
Foreign keys: none.
Referenced by:REFERENCE_DOCUMENTS

FORMATS_LOV: stores the possible formats (e.g. number formats, date formats) for value domains.
Foreign keys: none.
Referenced by:VALUE_DOMAINS

LANGUAGES _LOV: defines the possible languages.
Foreign keys: none.
Referenced by:DEFINITIONS, DESIGNATIONS, REFERENCE_DOCUMENTS

LIFECYCLES_LOV (LL): represents the lifecycle of a context.
Foreign keys: none.
Referenced by:CONTEXTS

OBJECT_CLASSES_LOV: represents the object classes of data element concepts.
Foreign keys: none.
Referenced by:DATA_ELEMENT_CONCEPTS

PERMISSIBLE_VALUES (PV): stores the allowed values for value domains.
Foreign keys: VD_PVS_HST
Referenced by: none.

PROGRAM_AREAS_LOV (PAL): represents the program area of a context.
Foreign keys: none.
Referenced by:CONTEXTS

PROPERTIES_LOV: represents the properties of data elements.
Foreign keys: none.
Referenced by:DATA_ELEMENTS

REFERENCE_DOCUMENTS: stores the reference documents of an administered component.
Foreign keys: ADMINISTERED_COMPONENTS, DOCUMENT_TYPES_LOV, LANGUAGES _LOV
Referenced by: none

RELATIONSHIPS_LOV: stores the possible types of relationships.
Foreign keys: none.
Referenced by: DEC_RECS, VD_PV_RECS, VD_RECS, DE_RECS, CSI_RECS, CS_RECS

UNIT_OF_MEASURES_LOV: stores values for units of measure (e.g. feet, miles, dollars, hours) for VDs.
Foreign keys: none.
Referenced by:VALUE_DOMAINS

VALUE_DOMAINS (VD): stores the value domain components, which encapsulate the sets of permissible values that can populate data elements.
Foreign keys: CONTEXTS, CONCEPTUAL_DOMAINS, AC_STATUS_LOV, DATATYPES_LOV, UNIT_OF_MEASURES_LOV, CHARACTER_SET_LOV, FORMATS_LOV

Referenced by: VD_RECS, VD_PVS, DATA_ELEMENTS.
Linked to: PERMISSIBLE_VALUES (VD_PVS)

VALUE_MEANINGS_LOV: stores the list of value meanings used by the CD_VMS and PERMISSIBLE_VALUES tables.
Foreign keys: none.
Referenced by:PERMISSIBLE_VALUES,CD_VMS

VD_PV_RECS: stores the relationships by which a permissible value is related to other permissible values.
Foreign keys: RELATIONSHIPS_LOV, VD_PVS
Referenced by: None

VD_PVS: associates value domains (VD) with their permissible values (PV).
Foreign keys: VALUE_DOMAINS,CONTEXTS, PERMISSIBLE_VALUES
Referenced by: VD_PV_RECS

VD_RECS: defines relationships (Part-of, Similar-To etc) between two value domains.
Foreign keys: VALUE_DOMAINS, RELATIONSHIPS_LOV
Referenced by: None

3.6 caDSR Entity Relationships

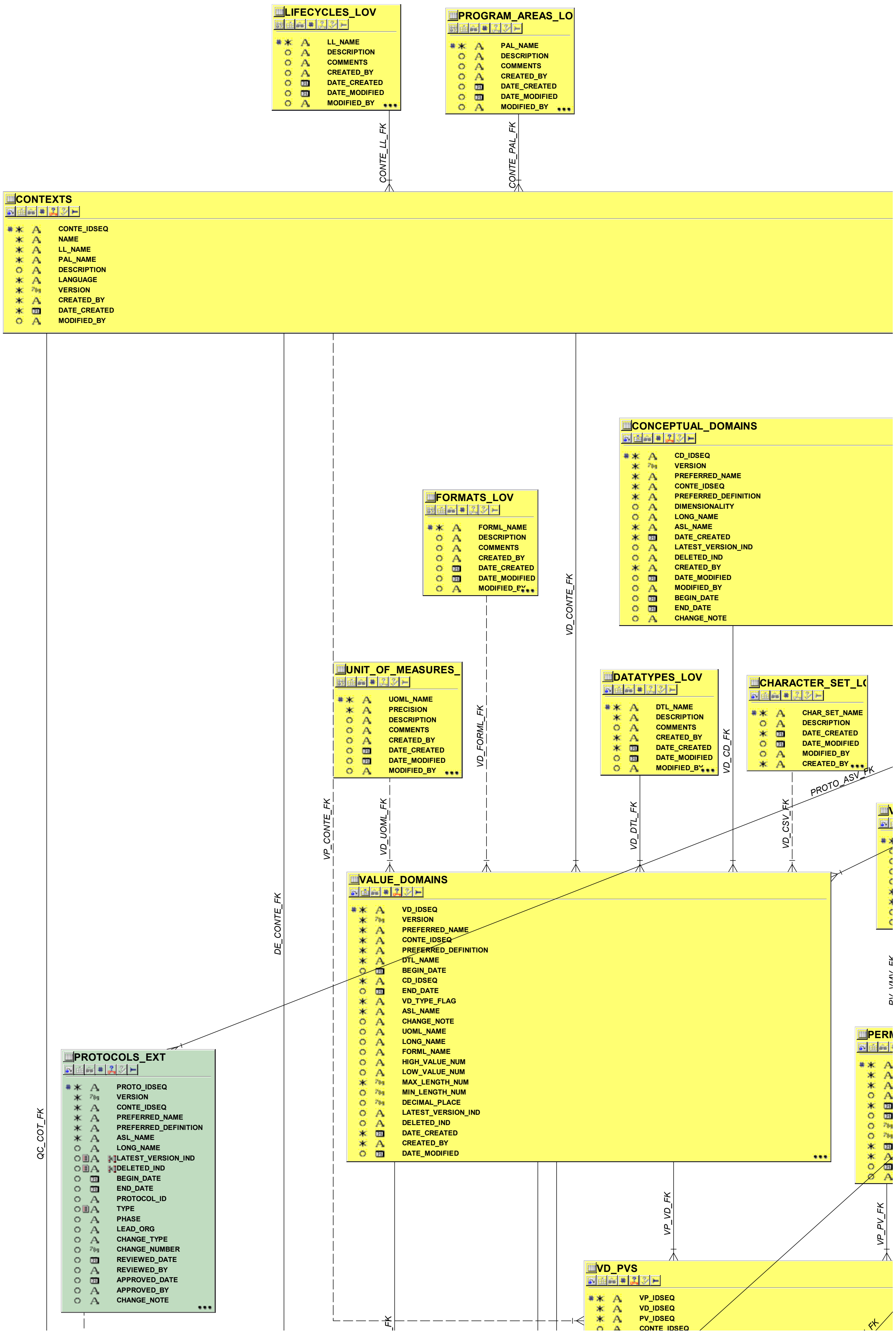
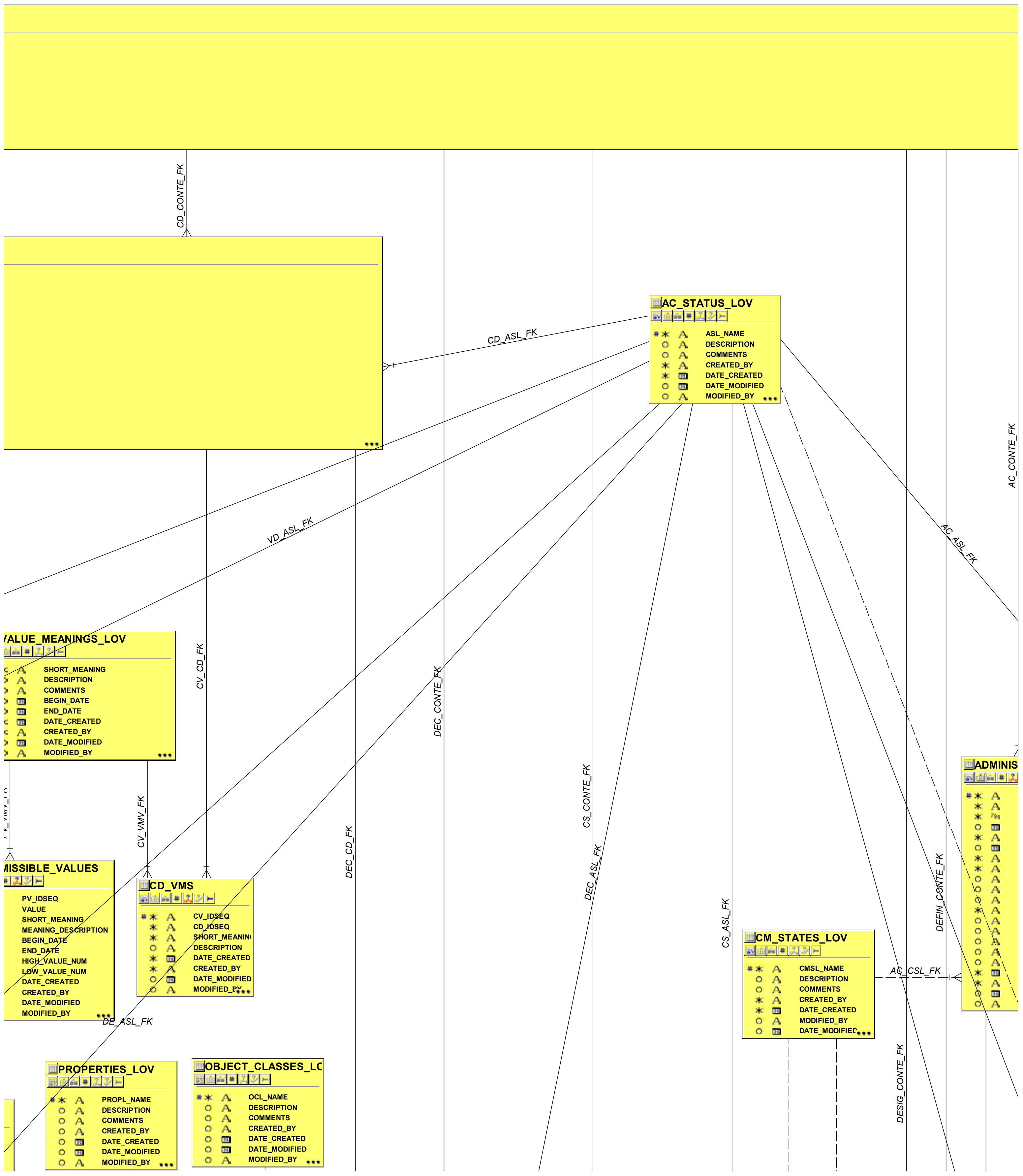
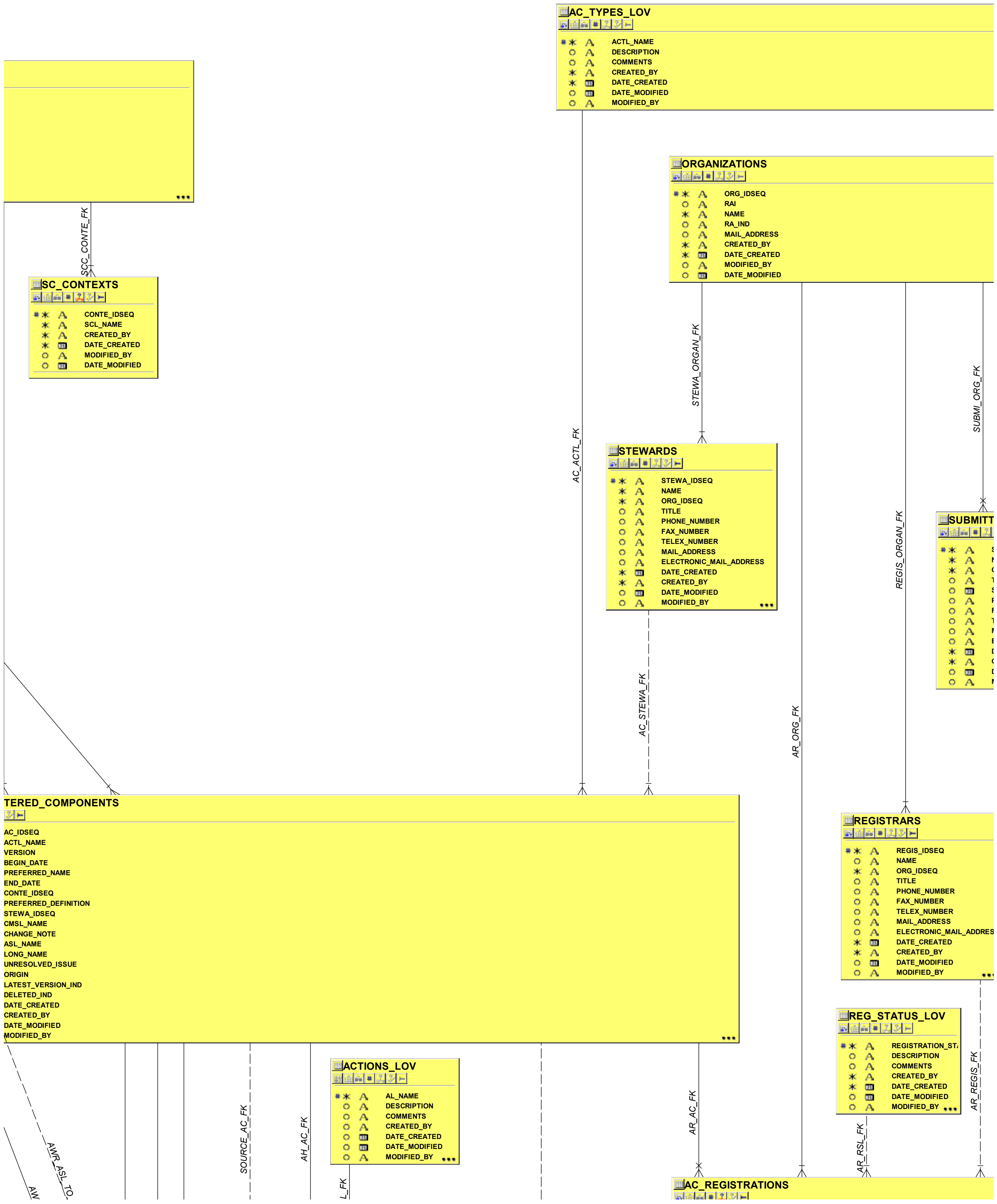
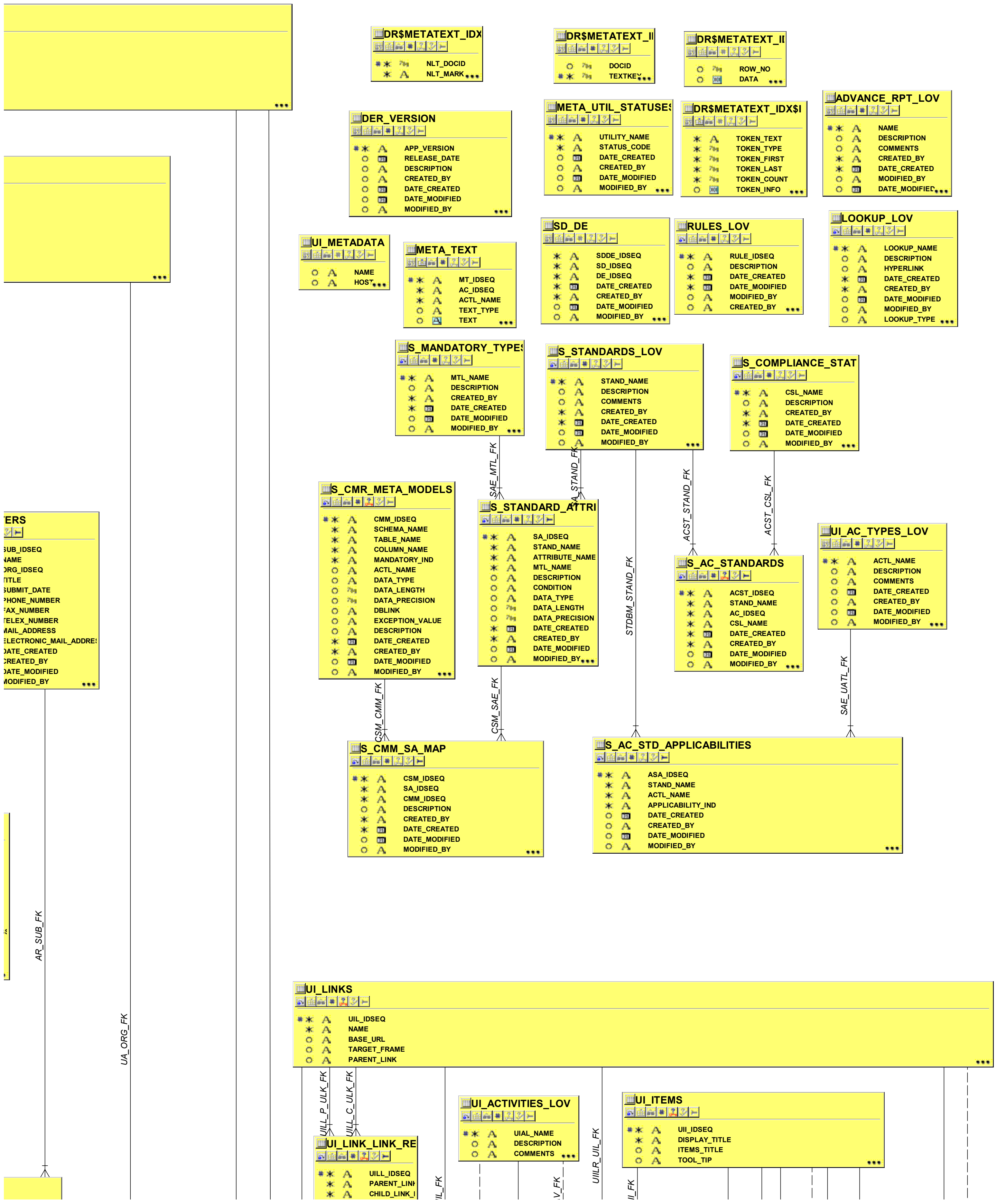
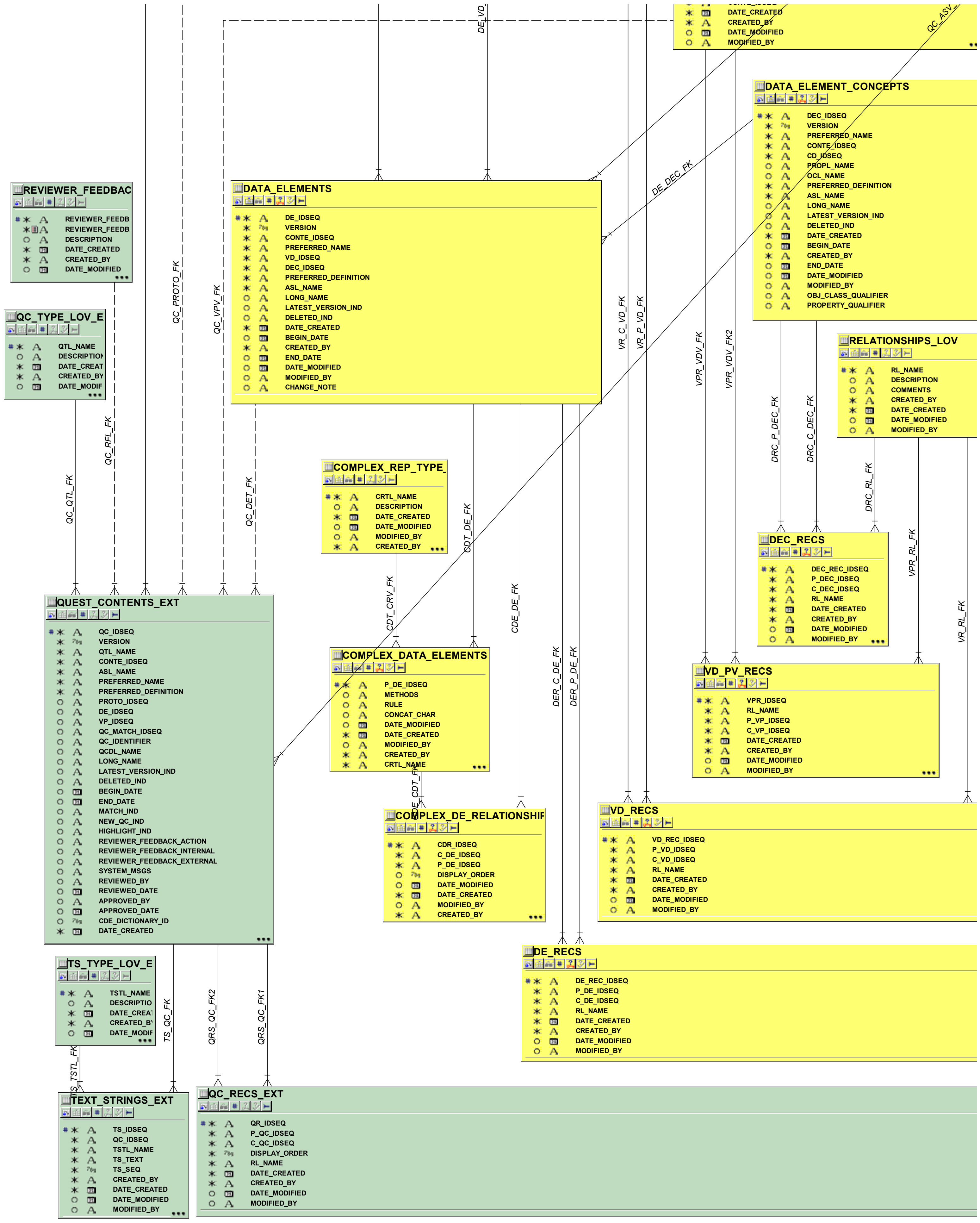

LIFECYCLES_LOV
LL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
PROGRAM_AREAS_LO
PAL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
CONTE_LL_FK
CONTE_PAL_FK
CONTEXTS
CONTE_IDSEQ
NAME
LL_NAME
PAL_NAME
DESCRIPTION
LANGUAGE
VERSION
CREATED_BY
DATE_CREATED
MODIFIED_BY
CONCEPTUAL_DOMAINS
CD_IDSEQ
VERSION
PREFERRED_NAME
CONTE_IDSEQ
PREFERRED_DEFINITION
DIMENSIONALITY
LONG_NAME
ASL_NAME
DATE_CREATED
LATEST_VERSION_IND
DELETED_IND
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
BEGIN_DATE
END_DATE
CHANGE_NOTE
FORMATS_LOV
FORML_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
VD_CONTE_FK
UNIT_OF_MEASURES_
UOML_NAME
PRECISION
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
VD_FORML_FK
DATATYPES_LOV
DTL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
VD_CD_FK
CHARACTER_SET_L
CHAR_SET_NAME
DESCRIPTION
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
CREATED_BY
PROTO_ASV_FK
VP_CONTE_FK
VD_UOML_FK
VD_DTL_FK
VD_CSV_FK
DE_CONTE_FK
VALUE_DOMAINS
VD_IDSEQ
VERSION
PREFERRED_NAME
CONTE_IDSEQ
PREFERRED_DEFINITION
DTL_NAME
BEGIN_DATE
CD_IDSEQ
END_DATE
VD_TYPE_FLAG
ASL_NAME
CHANGE_NOTE
UOML_NAME
LONG_NAME
FORML_NAME
HIGH_VALUE_NUM
LOW_VALUE_NUM
MAX_LENGTH_NUM
MIN_LENGTH_NUM
DECIMAL_PLACE
LATEST_VERSION_IND
DELETED_IND
DATE_CREATED
CREATED_BY
DATE_MODIFIED
PV_VMV_FK
PERM
QC_COT_FK
PROTOCOLS_EXT
PROTO_IDSEQ
VERSION
CONTE_IDSEQ
PREFERRED_NAME
PREFERRED_DEFINITION
ASL_NAME
LONG_NAME
LATEST_VERSION_IND
DELETED_IND
BEGIN_DATE
END_DATE
PROTOCOL_ID
TYPE
PHASE
LEAD_ORG
CHANGE_TYPE
CHANGE_NUMBER
REVIEWED_DATE
REVIEWED_BY
APPROVED_DATE
APPROVED_BY
CHANGE_NOTE
VP_VD_FK
VP_PV_FK
VD_PVS
VP_IDSEQ
VD_IDSEQ
PV_IDSEQ
CONTE_IDSEQ
FK
FK

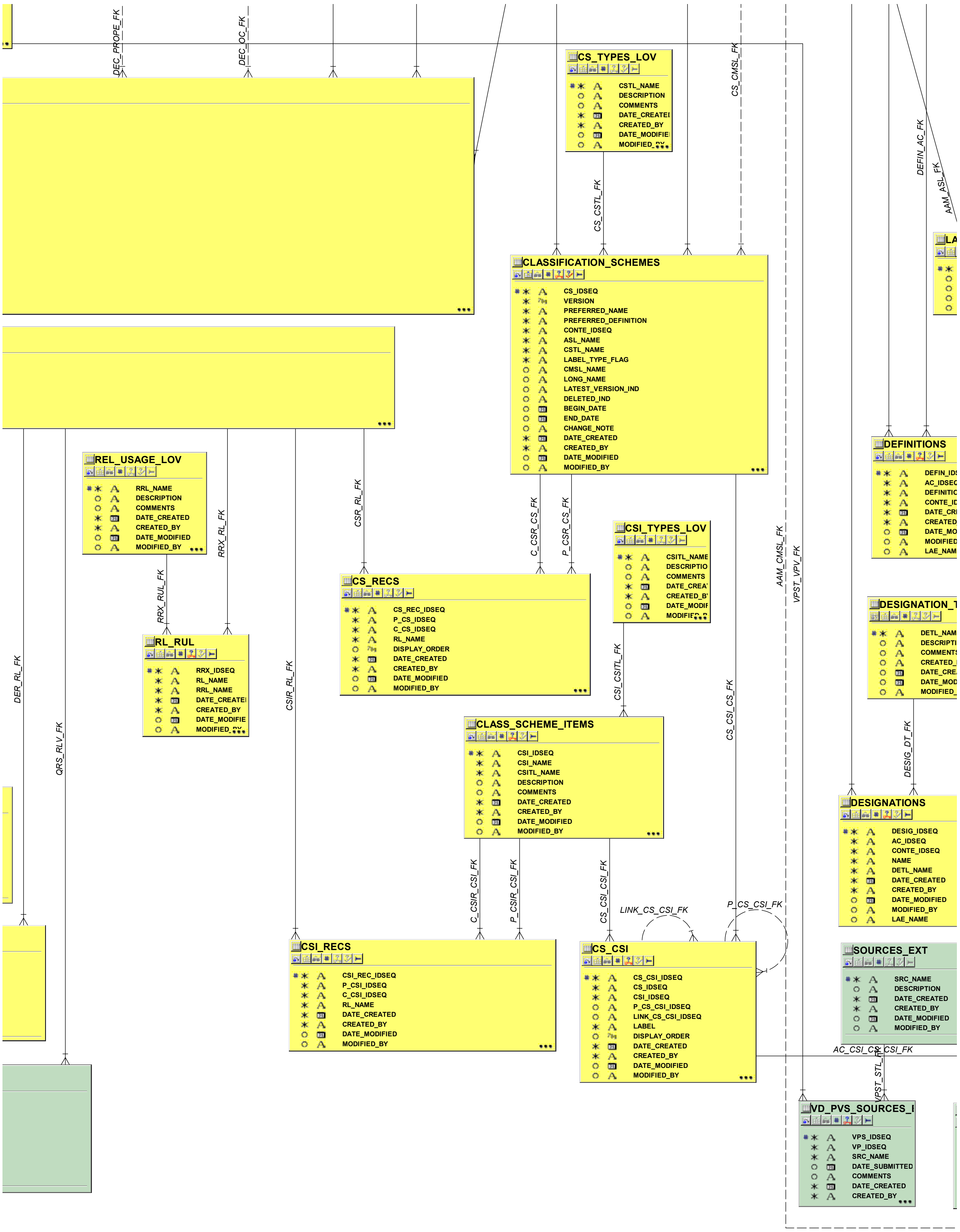
Diagram : DER 1.2.1 SERVER MODEL V3
Title : SBR V 1.2.1 REV Engineered
Modified : 16 October 2001 14:28:34
Author :
Application System : SBR v1.2.1 Reverse Engineered
Workarea : SBR Development

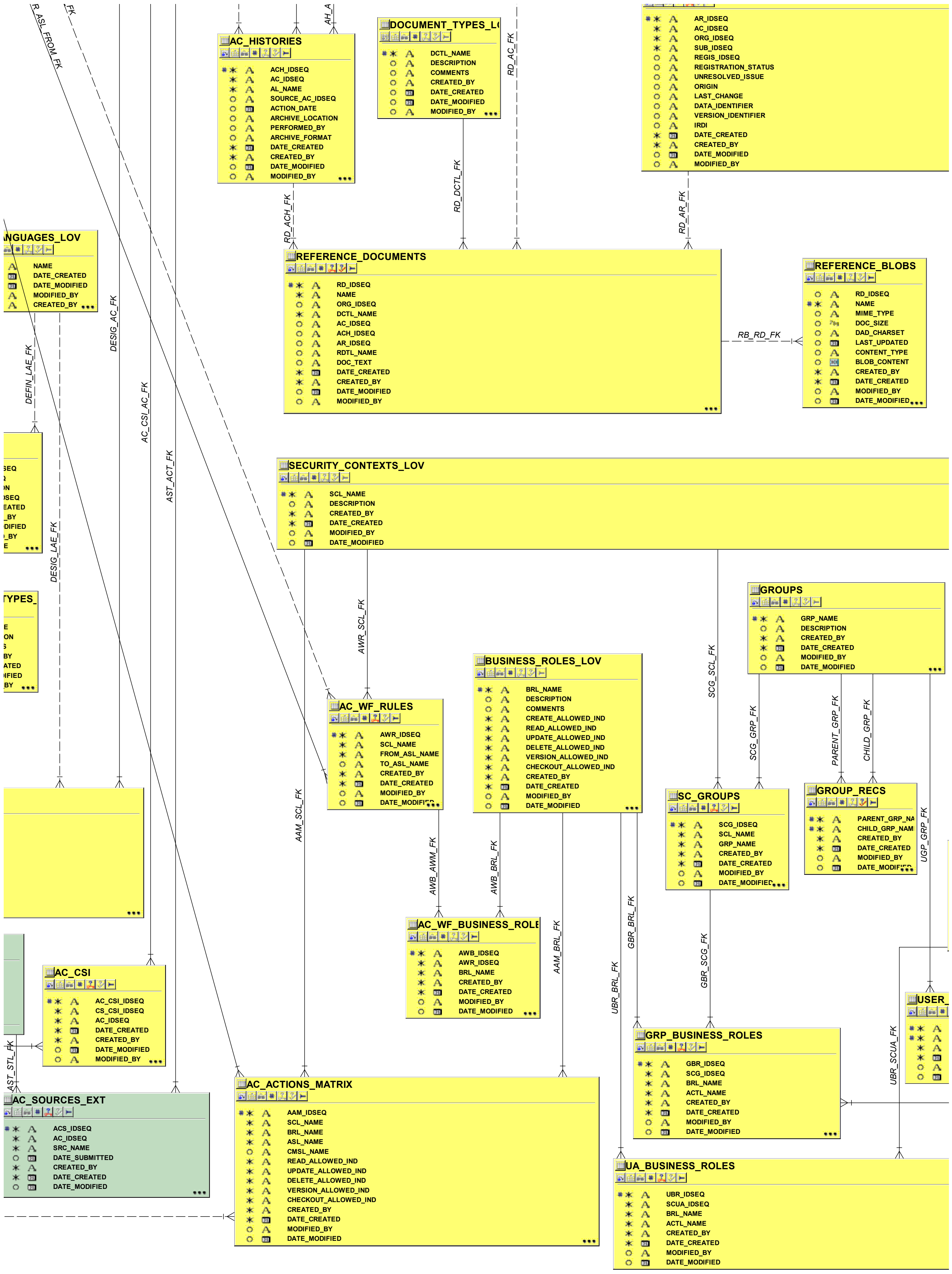


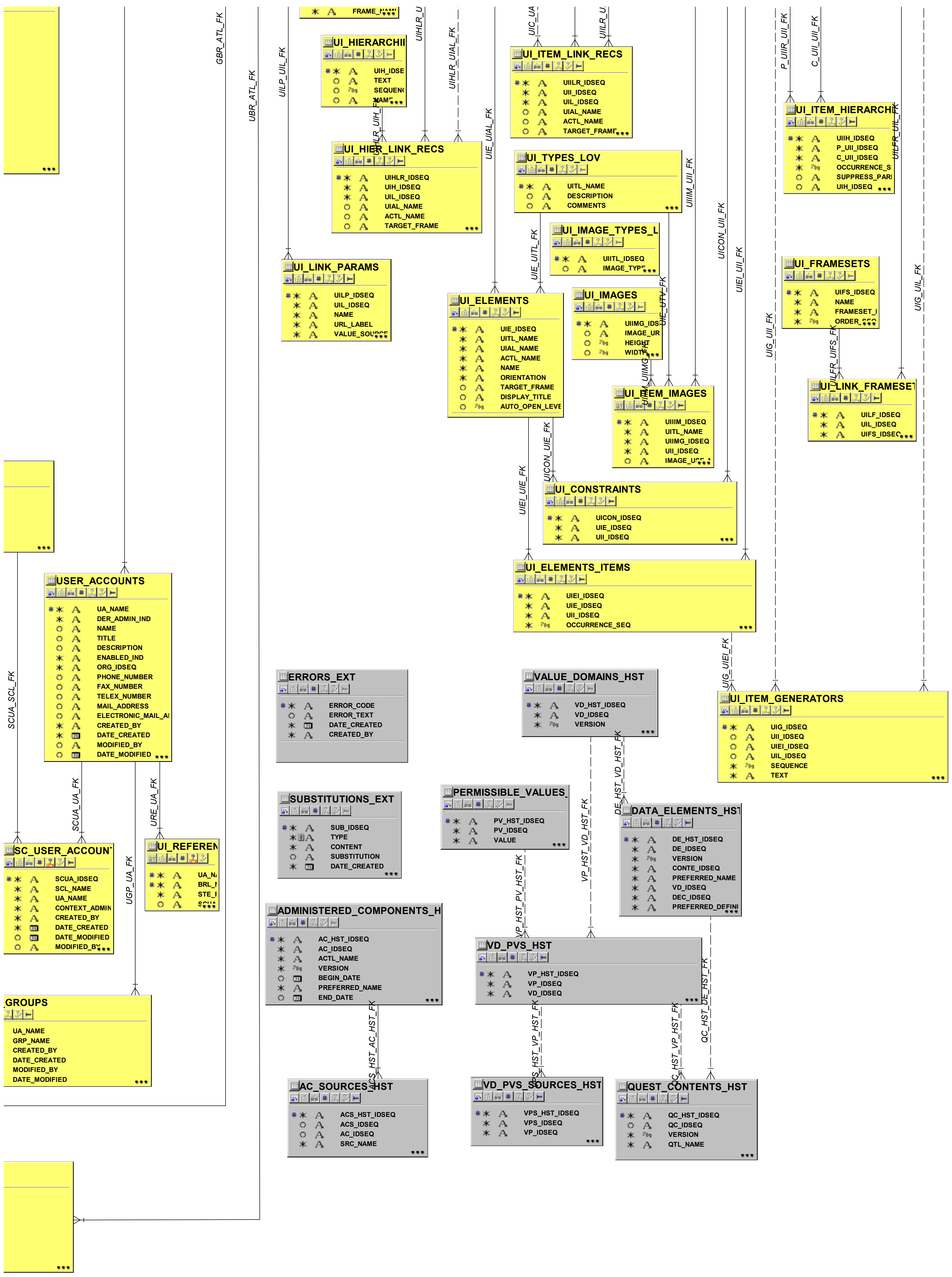

AC_TYPES_LOV
ACTL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
ORGANIZATIONS
ORG_IDSEQ
RAI
NAME
RA_IND
MAIL_ADDRESS
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
SCC_CONTE_FK
SC_CONTEXTS
CONTE_IDSEQ
SCL_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
STEWA_ORGAN_FK
SUBMI_ORG_FK
AC_ACTL_FK
STEWARDS
STEWA_IDSEQ
NAME
ORG_IDSEQ
TITLE
PHONE_NUMBER
FAX_NUMBER
TELEX_NUMBER
MAIL_ADDRESS
ELECTRONIC_MAIL_ADDRESS
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
SUBMITT
REGIS_ORGAN_FK
AC_STEWA_FK
AR_ORG_FK
TERED_COMPONENTS
AC_IDSEQ
ACTL_NAME
VERSION
BEGIN_DATE
PREFERRED_NAME
END_DATE
CONTE_IDSEQ
PREFERRED_DEFINITION
STEWA_IDSEQ
CMSL_NAME
CHANGE_NOTE
ASL_NAME
LONG_NAME
UNRESOLVED_ISSUE
ORIGIN
LATEST_VERSION_IND
DELETED_IND
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
REGISTRARS
REGIS_IDSEQ
NAME
ORG_IDSEQ
TITLE
PHONE_NUMBER
FAX_NUMBER
TELEX_NUMBER
MAIL_ADDRESS
ELECTRONIC_MAIL_ADDRES
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
REG_STATUS_LOV
REGISTRATION_ST/
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
ACTIONS_LOV
AL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
AR_REGIS_FK
AR_AC_FK
AR_RSL_FK
SOURCE_AC_FK
AH_AC_FK
AWR_ASL_TO
AW
L_FK
AC_REGISTRATIONS


DR$METATEXT_IDX
NLT_DOCID
NLT_MARK
DR$METATEXT_II
DOCID
TEXTKEY
DR$METATEXT_II
ROW_NO
DATA
DER_VERSION
APP_VERSION
RELEASE_DATE
DESCRIPTION
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
META_UTIL_STATUSES
UTILITY_NAME
STATUS_CODE
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DR$METATEXT_IDX$I
TOKEN_TEXT
TOKEN_TYPE
TOKEN_FIRST
TOKEN_LAST
TOKEN_COUNT
TOKEN_INFO
ADVANCE_RPT_LOV
NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
UI_METADATA
NAME
HOST
META_TEXT
MT_IDSEQ
AC_IDSEQ
ACTL_NAME
TEXT_TYPE
TEXT
SD_DE
SDDE_IDSEQ
SD_IDSEQ
DE_IDSEQ
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
RULES_LOV
RULE_IDSEQ
DESCRIPTION
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
CREATED_BY
LOOKUP_LOV
LOOKUP_NAME
DESCRIPTION
HYPERLINK
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
LOOKUP_TYPE
S_MANDATORY_TYPES
MTL_NAME
DESCRIPTION
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
S_STANDARDS_LOV
STAND_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
S_COMPLIANCE_STAT
CSL_NAME
DESCRIPTION
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
ERS
SUB_IDSEQ
NAME
ORG_IDSEQ
TITLE
SUBMIT_DATE
PHONE_NUMBER
FAX_NUMBER
TELEX_NUMBER
MAIL_ADDRESS
ELECTRONIC_MAIL_ADDRE
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
S_CMR_META_MODELS
CMM_IDSEQ
SCHEMA_NAME
TABLE_NAME
COLUMN_NAME
MANDATORY_IND
ACTL_NAME
DATA_TYPE
DATA_LENGTH
DATA_PRECISION
DBLINK
EXCEPTION_VALUE
DESCRIPTION
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
SAE_MTL_FK
SA_STAND_FK
S_STANDARD_ATTRI
SA_IDSEQ
STAND_NAME
ATTRIBUTE_NAME
MTL_NAME
DESCRIPTION
CONDITION
DATA_TYPE
DATA_LENGTH
DATA_PRECISION
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
STDBM_STAND_FK
ACST_STAND_FK
ACST_CSL_FK
S_AC_STANDARDS
ACST_IDSEQ
STAND_NAME
AC_IDSEQ
CSL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
UI_AC_TYPES_LOV
ACTL_NAME
DESCRIPTION
COMMENTS
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
CSM_CMM_FK
CSM_SAE_FK
SAE_UATL_FK
S_CMM_SA_MAP
CSM_IDSEQ
SA_IDSEQ
CMM_IDSEQ
DESCRIPTION
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
S_AC_STD_APPLICABILITIES
ASA_IDSEQ
STAND_NAME
ACTL_NAME
APPLICABILITY_IND
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
AR_SUB_FK
UA_ORG_FK
UI_LINKS
UIL_IDSEQ
NAME
BASE_URL
TARGET_FRAME
PARENT_LINK
UILL_P_ULK_FK
UILL_C_ULK_FK
UI_LINK_LINK_RE
UILL_IDSEQ
PARENT_LINI
CHILD_LINK_I
UI_ACTIVITIES_LOV
UIAL_NAME
DESCRIPTION
COMMENTS
UI_ITEMS
UII_IDSEQ
DISPLAY_TITLE
ITEMS_TITLE
TOOL_TIP
IL_FK
V_FK
UIILR_UIL_FK
IL_FK


DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DE_VD_
QC_ASV_
DATA_ELEMENT_CONCEPTS
DEC_IDSEQ
VERSION
PREFERRED_NAME
CONTE_IDSEQ
CD_IDSEQ
PROPL_NAME
OCL_NAME
PREFERRED_DEFINITION
ASL_NAME
LONG_NAME
LATEST_VERSION_IND
DELETED_IND
DATE_CREATED
BEGIN_DATE
CREATED_BY
END_DATE
DATE_MODIFIED
MODIFIED_BY
OBJ_CLASS_QUALIFIER
PROPERTY_QUALIFIER
DE_DEC_FK
REVIEWER_FEEDBAC
REVIEWER_FEEDB
REVIEWER_FEEDB
DESCRIPTION
DATE_CREATED
CREATED_BY
DATE_MODIFIED
QC_PROTO_FK
QC_VPV_FK
DATA_ELEMENTS
DE_IDSEQ
VERSION
CONTE_IDSEQ
PREFERRED_NAME
VD_IDSEQ
DEC_IDSEQ
PREFERRED_DEFINITION
ASL_NAME
LONG_NAME
LATEST_VERSION_IND
DELETED_IND
DATE_CREATED
BEGIN_DATE
CREATED_BY
END_DATE
DATE_MODIFIED
MODIFIED_BY
CHANGE_NOTE
VR_C_VD_FK
VR_P_VD_FK
VPR_VDV_FK
VPR_VDV_FK2
QC_TYPE_LOV_E
QTL_NAME
DESCRIPTION
DATE_CREAT
CREATED_BY
DATE_MODIF
RELATIONSHIPS_LOV
RL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
DRC_P_DEC_FK
DRC_C_DEC_FK
QC_RFL_FK
QC_QTL_FK
QC_DET_FK
COMPLEX_REP_TYPE_
CRTL_NAME
DESCRIPTION
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
CREATED_BY
CDT_DE_FK
DRC_RL_FK
DEC_RECS
DEC_REC_IDSEQ
P_DEC_IDSEQ
C_DEC_IDSEQ
RL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
VPR_RL_FK
CDT_CRV_FK
CDE_DE_FK
VR_RL_FK
QUEST_CONTENTS_EXT
QC_IDSEQ
VERSION
QTL_NAME
CONTE_IDSEQ
ASL_NAME
PREFERRED_NAME
PREFERRED_DEFINITION
PROTO_IDSEQ
DE_IDSEQ
VP_IDSEQ
QC_MATCH_IDSEQ
QC_IDENTIFIER
QCDL_NAME
LONG_NAME
LATEST_VERSION_IND
DELETED_IND
BEGIN_DATE
END_DATE
MATCH_IND
NEW_QC_IND
HIGHLIGHT_IND
REVIEWER_FEEDBACK_ACTION
REVIEWER_FEEDBACK_INTERNAL
REVIEWER_FEEDBACK_EXTERNAL
SYSTEM_MSGS
REVIEWED_BY
REVIEWED_DATE
APPROVED_BY
APPROVED_DATE
CDE_DICTIONARY_ID
DATE_CREATED
COMPLEX_DATA_ELEMENTS
P_DE_IDSEQ
METHODS
RULE
CONCAT_CHAR
DATE_MODIFIED
DATE_CREATED
MODIFIED_BY
CREATED_BY
CRTL_NAME
DER_C_DE_FK
DER_P_DE_FK
VD_PV_RECS
VPR_IDSEQ
RL_NAME
P_VP_IDSEQ
C_VP_IDSEQ
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
CDE_CDT_FK
COMPLEX_DE_RELATIONSHIP
CDR_IDSEQ
C_DE_IDSEQ
P_DE_IDSEQ
DISPLAY_ORDER
DATE_MODIFIED
DATE_CREATED
MODIFIED_BY
CREATED_BY
VD_RECS
VD_REC_IDSEQ
P_VD_IDSEQ
C_VD_IDSEQ
RL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DE_RECS
DE_REC_IDSEQ
P_DE_IDSEQ
C_DE_IDSEQ
RL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
TS_TYPE_LOV_E
TSTL_NAME
DESCRIPTIO
DATE_CREAT
CREATED_BY
DATE_MODIF
TS_QC_FK
QRS_QC_FK2
QRS_QC_FK1
TS_TSTL_FK
TEXT_STRINGS_EXT
TS_IDSEQ
QC_IDSEQ
TSTL_NAME
TS_TEXT
TS_SEQ
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
QC_RECS_EXT
QR_IDSEQ
P_QC_IDSEQ
C_QC_IDSEQ
DISPLAY_ORDER
RL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY


DEC_PROPE_FK
DEC_OC_FK
CS_TYPES_LOV
CSTL_NAME
DESCRIPTION
COMMENTS
DATE_CREATEI
CREATED_BY
DATE_MODIFIEI
MODIFIED_BY
CS_CMSL_FK
DEFIN_AC_FK
AAM_ASL_FK
CS_CSTL_FK
CLASSIFICATION_SCHEMES
CS_IDSEQ
VERSION
PREFERRED_NAME
PREFERRED_DEFINITION
CONTE_IDSEQ
ASL_NAME
CSTL_NAME
LABEL_TYPE_FLAG
CMSL_NAME
LONG_NAME
LATEST_VERSION_IND
DELETED_IND
BEGIN_DATE
END_DATE
CHANGE_NOTE
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DEFINITIONS
DEFIN_IDS
AC_IDSEQ
DEFINITIO
CONTE_ID
DATE_CRI
CREATED
DATE_MO
MODIFIED
LAE_NAM
REL_USAGE_LOV
RRL_NAME
DESCRIPTION
COMMENTS
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
RRX_RL_FK
CSR_RL_FK
C_CSR_CS_FK
P_CSR_CS_FK
CSI_TYPES_LOV
CSITL_NAME
DESCRIPTIO
COMMENTS
DATE_CREA
CREATED_B
DATE_MODI
MODIFIED_B
AAM_CMSL_FK
VPST_VPV_FK
DESIGNATION_T
DETL_NAM
DESCRIPTI
COMMENTS
CREATED_
DATE_CREA
DATE_MOD
MODIFIED_
RRX_RUL_FK
RL_RUL
RRX_IDSEQ
RL_NAME
RRL_NAME
DATE_CREATEI
CREATED_BY
DATE_MODIFIE
MODIFIED_BY
CS_RECS
CS_REC_IDSEQ
P_CS_IDSEQ
C_CS_IDSEQ
RL_NAME
DISPLAY_ORDER
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DER_RL_FK
CSIR_RL_FK
CSI_CSITL_FK
CS_CSI_CS_FK
QRS_RLV_FK
CLASS_SCHEME_ITEMS
CSI_IDSEQ
CSI_NAME
CSITL_NAME
DESCRIPTION
COMMENTS
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DESIG_DT_FK
DESIGNATIONS
DESIG_IDSEQ
AC_IDSEQ
CONTE_IDSEQ
NAME
DETL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
LAE_NAME
C_CSIR_CSI_FK
P_CSIR_CSI_FK
CS_CSI_CSI_FK
LINK_CS_CSI_FK
P_CS_CSI_FK
CSI_RECS
CSI_REC_IDSEQ
P_CSI_IDSEQ
C_CSI_IDSEQ
RL_NAME
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
CS_CSI
CS_CSI_IDSEQ
CS_IDSEQ
CSI_IDSEQ
P_CS_CSI_IDSEQ
LINK_CS_CSI_IDSEQ
LABEL
DISPLAY_ORDER
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
SOURCES_EXT
SRC_NAME
DESCRIPTION
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
AC_CSI_CS_CSI_FK
VPST_STL_FK
VD_PVS_SOURCES_I
VPS_IDSEQ
VP_IDSEQ
SRC_NAME
DATE_SUBMITTED
COMMENTS
DATE_CREATED
CREATED_BY


R_ASL_FROM_FK
FK
AH_A
AC_HISTORIES
ACH_IDSEQ
AC_IDSEQ
AL_NAME
SOURCE_AC_IDSEQ
ACTION_DATE
ARCHIVE_LOCATION
PERFORMED_BY
ARCHIVE_FORMAT
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
DOCUMENT_TYPES_L
DCTL_NAME
DESCRIPTION
COMMENTS
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
RD_AC_FK
AR_IDSEQ
AC_IDSEQ
ORG_IDSEQ
SUB_IDSEQ
REGIS_IDSEQ
REGISTRATION_STATUS
UNRESOLVED_ISSUE
ORIGIN
LAST_CHANGE
DATA_IDENTIFIER
VERSION_IDENTIFIER
IRDI
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
RD_ACH_FK
RD_DCTL_FK
RD_AR_FK
NGUAGES_LOV
NAME
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
CREATED_BY
DESIG_AC_FK
REFERENCE_DOCUMENTS
RD_IDSEQ
NAME
ORG_IDSEQ
DCTL_NAME
AC_IDSEQ
ACH_IDSEQ
AR_IDSEQ
RDTL_NAME
DOC_TEXT
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
RB_RD_FK
REFERENCE_BLOBS
RD_IDSEQ
NAME
MIME_TYPE
DOC_SIZE
DAD_CHARSET
LAST_UPDATED
CONTENT_TYPE
BLOB_CONTENT
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
DEFIN_LAE_FK
AC_CSI_AC_FK
AST_ACT_FK
SEQ
Q
ON
DSEQ
EATED
_BY
DIFIED
_BY
E
SECURITY_CONTEXTS_LOV
SCL_NAME
DESCRIPTION
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
DESIG_LAE_FK
TYPES_
E
ON
S
BY
ATED
DIFIED
BY
AWR_SCL_FK
SCG_SCL_FK
GROUPS
GRP_NAME
DESCRIPTION
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
BUSINESS_ROLES_LOV
BRL_NAME
DESCRIPTION
COMMENTS
CREATE_ALLOWED_IND
READ_ALLOWED_IND
UPDATE_ALLOWED_IND
DELETE_ALLOWED_IND
VERSION_ALLOWED_IND
CHECKOUT_ALLOWED_IND
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
AC_WF_RULES
AWR_IDSEQ
SCL_NAME
FROM_ASL_NAME
TO_ASL_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
SCG_GRP_FK
PARENT_GRP_FK
CHILD_GRP_FK
SC_GROUPS
SCG_IDSEQ
SCL_NAME
GRP_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
GROUP_RECS
PARENT_GRP_NA
CHILD_GRP_NAM
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
UGP_GRP_FK
AAM_SCL_FK
AWB_AWM_FK
AWB_BRL_FK
GBR_BRL_FK
AC_WF_BUSINESS_ROLE
AWB_IDSEQ
AWR_IDSEQ
BRL_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
AAM_BRL_FK
GBR_SCG_FK
UBR_BRL_FK
AC_CSI
AC_CSI_IDSEQ
CS_CSI_IDSEQ
AC_IDSEQ
DATE_CREATED
CREATED_BY
DATE_MODIFIED
MODIFIED_BY
USER_
UBR_SCUA_FK
AST_STL_FK
GRP_BUSINESS_ROLES
GBR_IDSEQ
SCG_IDSEQ
BRL_NAME
ACTL_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
AC_SOURCES_EXT
ACS_IDSEQ
AC_IDSEQ
SRC_NAME
DATE_SUBMITTED
CREATED_BY
DATE_CREATED
DATE_MODIFIED
AC_ACTIONS_MATRIX
AAM_IDSEQ
SCL_NAME
BRL_NAME
ASL_NAME
CMSL_NAME
READ_ALLOWED_IND
UPDATE_ALLOWED_IND
DELETE_ALLOWED_IND
VERSION_ALLOWED_IND
CHECKOUT_ALLOWED_IND
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
UA_BUSINESS_ROLES
UBR_IDSEQ
SCUA_IDSEQ
BRL_NAME
ACTL_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED


FRAME_NAME
UI_HIERARCHI
UIH_IDSEQ
TEXT
SEQUENC
NAME
UI_HIER_LINK_RECS
UIHLR_IDSEQ
UIH_IDSEQ
UIL_IDSEQ
UIAL_NAME
ACTL_NAME
TARGET_FRAME
UI_ITEM_LINK_RECS
UIILR_IDSEQ
UII_IDSEQ
UIL_IDSEQ
UIAL_NAME
ACTL_NAME
TARGET_FRAME
UI_TYPES_LOV
UITL_NAME
DESCRIPTION
COMMENTS
UI_IMAGE_TYPES_L
UIITL_IDSEQ
IMAGE_TYPE
UI_ITEM_HIERARCHI
UIIH_IDSEQ
P_UII_IDSEQ
C_UII_IDSEQ
OCCURRENCE_S
SUPPRESS_PAR
UIH_IDSEQ
UI_LINK_PARAMS
UILP_IDSEQ
UIL_IDSEQ
NAME
URL_LABEL
VALUE_SOURCE
UI_ELEMENTS
UIE_IDSEQ
UITL_NAME
UIAL_NAME
ACTL_NAME
NAME
ORIENTATION
TARGET_FRAME
DISPLAY_TITLE
AUTO_OPEN_LEVE
UI_IMAGES
UIIMG_IDS
IMAGE_UR
HEIGHT
WIDTH
UI_FRAMESETS
UIFS_IDSEQ
NAME
FRAMESET_I
ORDER_SEQ
UI_ITEM_IMAGES
UIIIM_IDSEQ
UITL_NAME
UIIMG_IDSEQ
UII_IDSEQ
IMAGE_URL
UI_LINK_FRAMESET
UILF_IDSEQ
UIL_IDSEQ
UIFS_IDSEQ
UI_CONSTRAINTS
UICON_IDSEQ
UIE_IDSEQ
UII_IDSEQ
UI_ELEMENTS_ITEMS
UIEI_IDSEQ
UIE_IDSEQ
UII_IDSEQ
OCCURRENCE_SEQ
USER_ACCOUNTS
UA_NAME
DER_ADMIN_IND
NAME
TITLE
DESCRIPTION
ENABLED_IND
ORG_IDSEQ
PHONE_NUMBER
FAX_NUMBER
TELEX_NUMBER
MAIL_ADDRESS
ELECTRONIC_MAIL_A
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
ERRORS_EXT
ERROR_CODE
ERROR_TEXT
DATE_CREATED
CREATED_BY
VALUE_DOMAINS_HST
VD_HST_IDSEQ
VD_IDSEQ
VERSION
UI_ITEM_GENERATORS
UIG_IDSEQ
UII_IDSEQ
UIEI_IDSEQ
UIL_IDSEQ
SEQUENCE
TEXT
SUBSTITUTIONS_EXT
SUB_IDSEQ
TYPE
CONTENT
SUBSTITUTION
DATE_CREATED
PERMISSIBLE_VALUES_
PV_HST_IDSEQ
PV_IDSEQ
VALUE
DATA_ELEMENTS_HST
DE_HST_IDSEQ
DE_IDSEQ
VERSION
CONTE_IDSEQ
PREFERRED_NAME
VD_IDSEQ
DEC_IDSEQ
PREFERRED_DEFINI
SC_USER_ACCOUN
SCUA_IDSEQ
SCL_NAME
UA_NAME
CONTEXT_ADMIN
CREATED_BY
DATE_CREATED
DATE_MODIFIED
MODIFIED_BY
UI_REFEREN
UA_N
BRL_I
STE_I
SCUA
ADMINISTERED_COMPONENTS_H
AC_HST_IDSEQ
AC_IDSEQ
ACTL_NAME
VERSION
BEGIN_DATE
PREFERRED_NAME
END_DATE
VD_PVS_HST
VP_HST_IDSEQ
VP_IDSEQ
VD_IDSEQ
GROUPS
UA_NAME
GRP_NAME
CREATED_BY
DATE_CREATED
MODIFIED_BY
DATE_MODIFIED
AC_SOURCES_HST
ACS_HST_IDSEQ
ACS_IDSEQ
AC_IDSEQ
SRC_NAME
VD_PVS_SOURCES_HST
VPS_HST_IDSEQ
VPS_IDSEQ
VP_IDSEQ
QUEST_CONTENTS_HST
QC_HST_IDSEQ
QC_IDSEQ
VERSION
QTL_NAME
GBR_ATL_FK
UBR_ATL_FK
UILP_UIL_FK
UIHLR_UIH_FK
UIHLR_UIAL_FK
UIH_FK
UIE_UIAL_FK
UIC_UA
UIILR_U
UIIM_UII_FK
UIE_UITL_FK
UIE_UTV_FK
UIIM_UIIMG_FK
UICON_UII_FK
UIEI_UII_FK
UICON_UIE_FK
UIEI_UIE_FK
P_UIIR_UII_FK
C_UII_UII_FK
UILFR_UIL_FK
UIG_UII_FK
UILFR_UIFS_FK
UIG_UIL_FK
UIG_UIEI_FK
SCUA_SCL_FK
SCUA_UA_FK
URE_UA_FK
UGP_UA_FK
DE_HST_VD_HST_FK
VP_HST_VD_HST_FK
VP_HST_PV_HST_FK
QC_HST_DE_HST_FK
ACS_HST_AC_HST_FK
VPS_HST_VP_HST_FK
QC_HST_VP_HST_FK